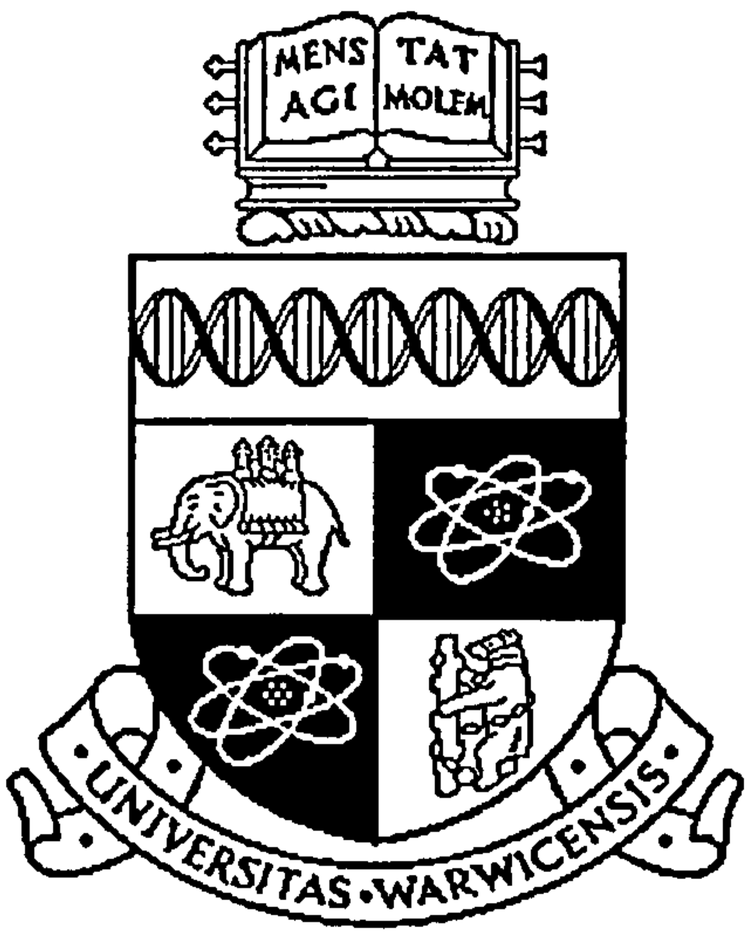
Development of a Standard Framework for Manufacturing Simulators

by

SUKHBINDER SINGH SANDHU

SUBMITTED FOR THE DEGREE OF Ph.D



at

THE UNIVERSITY OF WARWICK

in

THE DEPARTMENT OF ENGINEERING

July 1997

List Of Figures

TABLES

ACKNOWLEDGMENTS

I wish to thank Mr. Rajat Roy for his supervision and support during the work. In addition I wish to thank the members of the simulation team and my family for their support and assistance during the entire period of the work.

Declaration

None of the material in this thesis has been used in the submission of any degree or award.

S.S.Sandhu

ABSTRACT

Discrete event simulation is now a well established modelling and experimental technique for the analysis of manufacturing systems since it was first employed as a technique, much of the research and commercial developments in the field have been concerned with improving the considerable task of model specification in order to improve productivity and reduce the level of modelling and programming expertise required. The main areas of research have been the development of modelling structures to bring modularity in program development, incorporating such structures in simulation software systems which would alleviate some of the programming burden, and the use of automatic programming systems to develop interfaces that would raise the model specification to a higher level of abstraction. A more recent development in the field has been the advent of a new generation of software, often referred to as manufacturing simulators, which have incorporated extensive manufacturing system domain knowledge in the model specification interface.

Many manufacturing simulators are now commercially available, but their development has not been based on any common standard. This is evident in the differences that exist between their interfaces, internal data representation methods and modelling capabilities. The lack of a standard makes it impossible to reuse any part of a model when a user finds its

necessary to move from one simulator to another. In such cases, not only a new modelling language has to be learnt but also the complete model has to be developed again requiring considerable time and effort. The motivation for the research was the need for the development of a standard that is necessary to improve reusability of models and is the first step towards interchangability of such models.

A standard framework for manufacturing simulators has been developed. It consists of a data model that is independent of any simulator, and a translation module for converting model specification data into the internal data representation of manufacturing simulators; the translators are application specific, but the methodology is common and illustrated for three popular simulators. The data model provides for a minimum common model data specification which is based on an extensive analysis of existing simulators. It uses dialogues for interface and the frame knowledge representation method for modular storage of data. The translation methodology uses production rules for data mapping.

ABBREVEATIONS

AI	Artificial Intelligence
AGV	Automatic Guided Vehicle
AS	Activity Scanning
CRMS	Common Representation for Manufacturing simulators
DES	Discrete Event Simulation
DOS	Disc Operating System
EDD	Earliest Due Date
ES	Event Scheduling
FIFO	First In First Out
FMS	Flexible Manufacturing Systems
GUI	Graphical User Interface
PI	Process Interaction
SMSF	Standard Manufacturing Simulator Framework
STEP	STandard for Exchange of Product model data
WIP	Work In Progress

CHAPTER 1. INTRODUCTION

As manufacturing systems become more complex the greater the effort required to plan, operate and improve them. Discrete Event Simulation (DES) is a dynamic analysis technique which is extensively used in the evaluation of new or existing manufacturing systems, providing support for strategic, tactical and operational planning. It provides an insight into the dynamic behaviour of the system resulting from the variability in behaviour of its components and interactions between them. It is a useful tool employed by managers and analysts responsible for the design and operation of complex manufacturing systems, for making informed investment decisions about new plants and facilities. It allows experimentation with different plant configurations and operating parameters before a decision is made on their implementation.

Simulation can be used by decision makers to try out ideas on a computer before building a new system or changing the operation of a system. If the computer program is a relevant and an accurate representation of the system, then experiments can be performed to predict what will happen in the manufacturing system under different operating conditions, allowing ideas to be evaluated that would be costly, disruptive or impossible to try out on the real system. For example, if a company was envisaging an extension to an existing facility and was uncertain as to its potential benefits, simulation

could be used to study the plant as it currently exists and as it would with the extension, to determine the effect. In addition, for existing facilities it can be used for evaluating schedules, new policies, decision rules, organisational structures and information flow without system disruption. From the operational perspective, without simulation it would be difficult to predict the effects of congestion caused by changes in production schedule, batch sizes, machine breakdowns and labour performance variability.

In the early days the use of simulation was restricted to the off-line analysis of complex systems on a mainframe and required the employment of an analyst with a good knowledge of the system and adequate simulation training. However, with the advent of PCs and development of a new generation of simulation systems its use has proliferated. A variety of people with knowledge of the real world system, but with little or no specialist training in simulation, are increasingly being involved in the validation of models, experimentation and, in many cases, as developers of models.

1.1 Early developments in Simulation

Model development is a time consuming and difficult task and much of the research in simulation has been concerned with simplifying the task by the inclusion of domain knowledge, simulation expertise and easier interfaces in simulation systems. However, up until about 1960, before the advent of simulation languages, models were developed in general purpose computer

languages like FORTRAN. These offered the greatest flexibility, but required all tasks in a simulation model to be programmed in a 3rd generation computer language making the model development process extremely labour intensive. This had the disadvantage of requiring an expert in programming and simulation for their development.

The first simulation language to be developed in the UK was GSP, from which followed other general purpose simulation languages like ECSL and SIMON since the early 1960's. The development of these general purpose simulation languages alleviated some of the model development burden by providing facilities for random number generation, sampling from probability distributions, automatic collection of output, constructs for modelling various elements usually associated with simulation modelling, etc. They still required the writing of a simulation program, but this was considerably easier than in a general purpose computer language.

The American research effort also resulted in the development of a number of general purpose simulation languages like GPSS, GASP II, SIMSCRIPT and SIMAN/CINEMA. A diagram, based on the one given in Carrie (1988), of an historical perspective in the development of some of the main simulation software, in the UK and USA, is given in Fig 1.

A major improvement evident in more recent versions of these languages was the inclusion of an animation facility, which allowed the visual depiction of the

modelled system. This allowed the status of resources to be observed as they interacted with materials and, in the process, aided debugging, validation and presentation. Animation, although no substitute for exhaustive model validation and statistical analysis, allowed validation to be carried out by a wider audience with little knowledge of simulation but an in-depth understanding of the system being modelled.

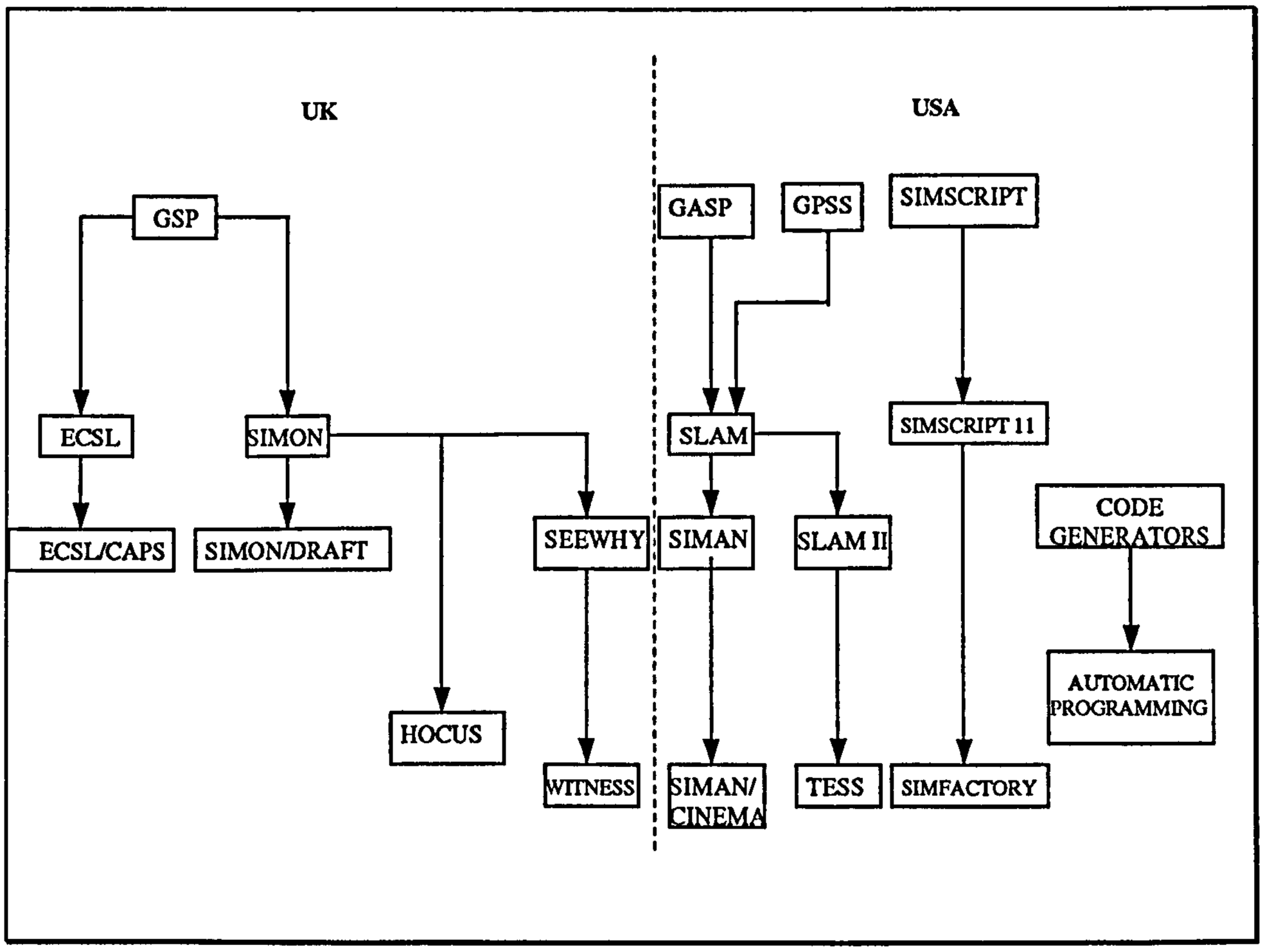


Fig 1 The historical development of some of the main Simulation Software.

A research trend that became evident from the mid 60's onwards was attempts at making the model specification easier, via some form of Automatic Programming. Automatic Programming is concerned with making program

specification easier by raising it from the programming to a higher more natural level. The early attempts at automatic programming for simulation resulted in simulation code generators, which were provided as front ends to existing general purpose simulation languages and contained only knowledge of the syntax of the target simulation language. They did not contain general simulation modelling knowledge, which had to be provided by the user in some form of high-level descriptor language. They can be effectively considered as simulation language statement generation tools. They were usually written in procedural languages like FORTRAN or Pascal, and the specification method was usually a questionnaire or interrogation of the user. Examples of code generators are CAPS and DRAFT for generating models in ECSL and SIMON respectively

An important development in simulation systems in the early 1980's was the advent of SEEWHY as a Visual Interactive Simulation (VIS) system which allowed the simultaneous creation of the simulation model and animation. VIS provides a means of fine tuning the validity of the simulation without having to perform a full length simulation run. In these Visual Interactive Systems icons are provided which may be linked together in the form of a network to depict the physical entities. This form of Visual Interactive Simulation was made possible by advances in graphics and animation and its advent was deemed the most important advance in simulation since the development of general purpose simulation languages.

Also, another important impact on simulation modelling became evident, even though the official techniques may not have been used, which was the incorporation of Artificial Intelligence.

1.2 The advent of AI

Artificial Intelligence (AI) has extended many of the frontiers of computer science; it has had uses in pattern recognition, speech processing, robotics, expert systems, language processing, etc, and is a means of creating methods of programming closer to human thinking compared to the traditional programming methods. Its main objective is to capture in a computer program behaviour which could be considered intelligent. AI attempts to simulate human intelligence via programming with rules, logic, neural nets or communicating objects.

There has always been an important link between Simulation and AI because, although using different representations, they attempt to model via a piece of software some aspects of the real world. The integration of the two was motivated by the increasing complexity of simulation models making it desirable to have a new generation of simulation systems which possess certain on-line intelligence to assist both the developers and the users. This has encouraged modelling based on inferencing, search methods and knowledge representation schemes developed in AI.

The main impact of AI has been on attempts at automatic modelling systems, instead of just simulation code generators, and the appearance of Knowledge Based Simulation Environments. The automatic modelling systems, due to improved inferencing and knowledge representational methods, enable a wider variety of simulation expertise to be represented and applied. They contain general simulation modelling and domain knowledge in addition to simulation target language knowledge, so that the model specification could be raised to an even higher level of abstraction. They also provide more advanced specification methods like natural language processors, graphics interfaces and multi-entry dialogue interrogations.

The Simulation Environments have also been the result of breakthroughs in AI resulting in improved interfaces, knowledge representation, and relational and object oriented databases These store simulation modelling knowledge using rules, frames, semantic nets, and object oriented programming. They use a variety of the specification methods used by the automatic programming and modelling systems. The main aim is not merely to produce code like automatic simulation programming, but to support the entire simulation model development life cycle from model building, validation and verification through to experimentation and statistical analysis.

Allied to the incorporation of AI was the impact of advances in software engineering research which resulted in improved user interfaces.

1.3 Developments in user interfaces

There has also been an improvement in the interfaces used for specifying a simulation model. In the early general purpose simulation languages like GPSS, SIMSCRIPT and CSL, the models were specified by using a text editor to enter a number of program statements. This process required the user to have an in-depth knowledge of the syntax of the simulation language. This problem was alleviated with the advent of higher level specification methods using Activity Cycle Diagrams (ACD), block diagrams (e.g. Blocks facility in SIMAN), etc, which can be thought of as statement generators or a form of code generator. This removed the need for the user to know about the syntax of a particular simulation language statement.

An important implication for simulation systems, especially Generic Simulators was the emergence of Windowing Environments incorporating on-line help. These allowed different parts of a model to be created and displayed in different windows via a combination of menus, dialogue boxes and graphics. The graphical interface could be used for creating the animation whilst dialogues could be used for entering the simulation model parameters. The dialogue boxes are used to provide contextual information to the user to:

- make a related set of choices.
- type in some information.
- choose from a set of options.

- acknowledge a piece of information before proceeding.

The entry of model parameters was further simplified in dialogue boxes using buttons, scroll boxes, dials or text fields with facilities for guiding the user and simultaneous error checking to ensure correct input.

1.4 Simulators

In order to further simplify the task of model specification in simulation, data driven models or special purpose simulators were developed, e.g. Mast, Rensam, Map/1, etc, which are preprogrammed simulation models specific to a range of systems, and require no programming on the part of the user. These are restricted to specific problem domains for example MAST and MAP/1 are intended for modeling Flexible Manufacturing Systems (FMS). In such systems the user simply provides the data which may be numerical (e.g. number and capacity of a machine), logical (e.g. a dispatching rule for an AGV) or textual.

The next stage in the evolution of simulation software lead in the early and mid 1980's to the development of generic simulators like WITNESS, PROMODEL, SIMFACTORY and FACTOR/AIM which segregated programming and data. These attempted to extend the flexibility of simulators to cover the entire manufacturing domain. Research into these Generic Manufacturing Simulators has proceeded in parallel with that into automatic

simulation programming, making the development and the use of the latter effectively redundant. These simulators which used the improved user interfaces, attempted to have greater flexibility without diminishing their ease of use, combining some of the advantages of general purpose simulation languages, more advanced interfaces and special purpose simulators. The development of these generic data driven languages, which separate data and programming, has made the model specification easier than with general purpose simulation languages without losing significant flexibility. They do not require the need for a sequential or procedural simulation program to be written and are designed to model systems which have a similar underlying structure. They reduce model development time, increase accuracy, improve communication and are predominately Visual Interactive.

The underlying structure in such systems is motivated by the realisation that manufacturing systems that are simulated have the same characteristics, but differ in details. For example, since all manufacturing systems comprise products, and resources to produce them such as facilities, machines, operators, handling devices, storage, pallets, fixtures, tools, etc, a lot of the research has been concerned with developing systems which are applicable to a variety of manufacturing systems with little modification.

This is particularly true for the generic simulators which include domain knowledge of a variety of manufacturing systems, providing pre-defined modelling constructs for real world objects whose characteristics can be easily

instantiated for a specific application. This is true even though they have followed different development paths, e.g. WITNESS (Istel, 1995) was initially intended for modeling production lines, PROMODEL (Production Modelling Corp, 1991) for systems with a family of parts and FACTOR/AIM (Pritsker Corp, 1992) for machine shops.

1.5 Research Objectives.

Automatic modelling systems (as well as code generators) drew a distinction between a specification language (interface) and the target simulation language. Like automatic modelling systems, manufacturing simulators also raised the level of specification by incorporating domain and simulation modelling knowledge in the interface, but did not maintain a distinction between the specification language and the simulation modelling language; the interface effectively acts as both.

Many manufacturing simulators are now commercially available, and their use in industry has rapidly increased, often by non-specialist manufacturing engineers. However, their development has not been based on any common standard, and this is evident in the differences that exist between their interfaces, internal data representation methods, modelling capabilities and, to a lesser extent, even in the way the simulation executive operates. A lack of a standard makes it impossible to reuse any part of a model when a user finds it necessary to move from one simulator to another; in such cases, not only a

new language has to be learnt but also the complete model has to be developed again, requiring considerable time and effort.

The objective of this research programme was to develop a standard framework for manufacturing simulators which would help with the reusability of models, or large parts of them between different simulators. The motivation is similar to the STEP research programme (Burkett and Yang. 1992)on the development of a standard for CAD system. The tight coupling of the model specification and simulation modelling in manufacturing simulators makes this a difficult task; a similar problem was also faced in STEP, and led to the separation of specification data model from the target modelling languages with the help of application referrence models (translators). The approach is similar to that taken in the automatic programming techniques but, unlike examples of the latter in the literature, development of standards cannot be based on specific target languages. Due to the highly logic intensive nature of simulation modelling techniques, it was also envisaged that complete standardisation is unlikely, and the development of a minimum standard was a more realistic objective. This minimum standard required the scope of the manufacturing systems covered by the standard to be set to covering the majority of automated manufacturing systems i.e mass production, job shops, batch and FMS. Also, within this scope the minimum standard was restricted to the structural elements and the process plan (the only behavioural element) of these systems.

The steps taken in this study towards the development of a standard framework can be summarised as:-

1. investigate the development of simulation software and their specification methods, in particular, advances in knowledge representation techniques, interfaces and automatic programming methods.
2. investigate and identify a minimum modelling standard for generic manufacturing simulators.
3. develop a data model for the standard model specification.
4. develop a translation method for creating (partial) simulation models in target manufacturing simulators from the data model, and illustrate with examples.

CHAPTER 2. The Development of Discrete Event Simulation

2.1 Introduction

In this chapter the nature of discrete event simulation and its main developments over the years are discussed. Since problem specification in a computer useable form is a time consuming and difficult task, much of the research in simulation has been concerned with simplifying it by the inclusion of modelling structures, domain knowledge, simulation expertise in the model specification interface, and the development of easier to use user interfaces. Much of the early work was concerned with the development of formal, general modelling structures (often referred to as world views), while later work concentrated on developing higher level modelling constructs containing domain knowledge. There has also been considerable effort in the improvement of the software environment, from the early days of models in 3rd generation general purpose languages (e.g. FORTRAN) to the present day generic simulators (e.g. WITNESS).

2.2 Application of Simulation

As manufacturing systems become more complex, the greater the effort required to plan, operate and improve them. Discrete Event Simulation (DES) is now a well established technique for strategic analysis and design

of manufacturing systems, and is increasingly being used as well in tactical and operational planning. DES involves the modelling of a system in such a manner that the model mimics the response of the actual system to a high (or desired) level of accuracy. Its importance to manufacturing industry is evident from the large number of commercial tools that have been developed in recent years particularly for this market.

DES allows effective representation of all important system components, together with their characteristics and interactions, and the inclusion of the variability in their behaviour. Whatever detail the modeller requires to make appropriate judgments about a system can usually be included, thus providing an environment for the realistic experimentation with system design alternatives and their operations.

Simulation can be used to evaluate the design of a proposed system (Ranky, 1983; Musselman, 1984; Schriber, 1987; Carrie, 1988) prior to its physical implementation; hence, it can be thought of as a forecasting technique, effectively allowing the users to operate the (virtual) system before its actual implementation. This is extremely useful since the cost of simulation is a fraction of the cost of rectifying faulty system design, making its use critical when the cost of the proposed system is high. When a system already exists, simulation can be used for studying ways of improving the design or investigating different operating policies and scheduling strategies without disturbing its operation in any way.

In addition to its traditional role as a design tool, DES is increasingly being employed as an aid to the creation of shop floor schedules (Kiran and Smith. 1983; Novels and Wichmann, 1990), which can be used to control the production activities of the real system. Through the execution of a detailed simulation model a finite capacity schedule is generated. The model includes the process routes of jobs and the resources they require, the initial conditions of the current shop floor status, known machine breakdowns, critical resources etc. Additionally scheduling rules can be included for assigning part allocation priorities, machine conflict resolution, labour allocation, tool allocation, etc. The event schedule that the simulation run generates can be given to the shop-floor as a realistic schedule consisting of a planned time ordered set of activities. However, before the schedule can be released to the shop floor, it has to be analysed to see whether it is acceptable in terms of the expected level of performance on, for example, due date satisfaction, throughput or utilisation of resources. If the schedule is deemed unacceptable a series of experiments may need to be conducted to refine it by increasing capacity (through overtime), reducing load (e.g. through sub-contracting of work), changing the priorities of critical jobs, etc. In recent years, a number of simulation based scheduling systems have appeared like Provisa (Istel, 1993) and INORDA (Insight, 1993). These provide useful facilities for scheduling such as built in scheduling rules and links to MRP systems.

Simulation is extremely useful in gaining confidence of non-technical clients because the results from a simulation are easier to understand than those from an analytical approach. This is especially true since the appearance of animation facilities within simulation software for displaying the movement of entities through a visual display. An important effect of this increased ease of

communication between the model builders and users is that the user can become involved in the model development process.

As a tool for system design and operation, DES also has a number of limitations. There is the need for specialised training in constructing simulation models, since manufacturing engineers are very seldom skilled in simulation techniques. Considerable time and effort are needed for data collection, model building, verification and validation, execution and the interpretation of results. Simulation is only an experimental tool and not an optimisation method; only a relatively small number of alternatives can be investigated in a reasonable time. Simulation experiments are also of a statistical nature and therefore require interpretation. However, some of these problem, in particular model development, have been addressed to some extent in recent years through a number of developments in the field, e.g. with the advent of Manufacturing Oriented Simulation Languages and Integrated Simulation Support Systems.

Although there is no all encompassing methodology which can be adopted to ensure a successful simulation project, it has been found from experience that a simulation project should proceed through a number of stages (Law and Kelton. 1982; Schriber, 1987; Carrie, 1988; Pidd, 1988). The stages are:

1. Understanding and definition and of the objectives, and specification of key issues to be addressed by the model. The objective(s) include: evaluation of a proposed design against a specific set of criteria; comparison of a number

of competing systems; development of operating policies or procedures; prediction of a system's performance under a specific set of conditions; identification of bottlenecks etc.

2. Model formulation phase, in which a conceptual model is envisioned to represent the system under study. The conceptual model is the model formulated in the mind of the modeller, identifying the basic building blocks of the system in terms of the manufacturing functions.
3. Model building to translate the conceptual model into a computer executable simulation model. This model may be defined in a 3rd generation high level language, general purpose simulation modelling language, special purpose package or a simulator.
4. Collection of the data required for the model; the extent of the task may be determined by the desired detail and accuracy of the model. The data required in a manufacturing simulation model include processing times, set-up times, failure times, repair times, transporter travel times, type of setup., etc.
5. Verification to examine the computer code to ensure that it conforms to the user's conceptual model of the real system.
6. Validation to determine whether the simulation model behaves in the manner of the real system and concentrates on the degree of fit between the real world and its model representation.
7. Experimental design to determine the correct combination of system variables that result in the best possible performance characteristics.
8. Model execution to provide results.

9. Statistical analysis to extract information from the experimental results.

2.3 Concepts of flexibility and ease of use

Concepts of flexibility and ease of use are important in any discussion on simulation methodology and software because they define the suitability of a simulation system to the problem domain, and the level of expertise required of the personnel who will develop and use the models.

Ease of use is an important and desirable feature in simulation software because it affects the speed of model development; determined by the method and tools available to the user in specifying the model. In the early systems there was very little assistance; models were specified with the aid of primitive text editors and by the writing of program statements in 3rd generation general purpose computer languages. The situation has improved with the advent of general purpose simulation languages, intelligent editors, simulators and Graphical User Interfaces (GUI).

Flexibility is an equally important feature in simulation software because no two manufacturing systems are the same. If the simulation software doesn't possess the necessary flexibility, then the model may have to be distorted to fit a particular system, which may result in a model of dubious or unknown accuracy. Alternatively the modeller may have to spend a great deal of time and effort in finding ways of overcoming the limitations of the particular

system. Flexibility is concerned with the ability the simulation software provides for modelling a wide range of different systems. A simulation software system is deemed highly flexible if it can be used to model a wide variety of systems with the inclusion of a substantial amount of the detail required by the modeller, whereas an inflexible system is one in which the amount of detail that can be included is restricted by the structure and the input requirements of the software.

It has been found from experience, that as the development of simulation software has progressed, increased ease of use has generally been at the expense of lower flexibility. This relationship between the ease of use and flexibility is illustrated in Fig 2.1 (based on an analysis by (Mills and Talavage, 1985)), which indicates that the special purpose simulators (MAST, GCMS, etc.) are the most easy to use (due to their entirely data-driven approach), but are limited in flexibility because they were designed to model only a specific type of manufacturing system. In contrast more programming oriented (as opposed to data-driven) simulation software have more flexibility but less ease of use. WITNESS (Istel Ltd, 1995) can be seen as an attempt to strike a balance between the two extremes via the inclusion of a programming language within a data-driven framework.

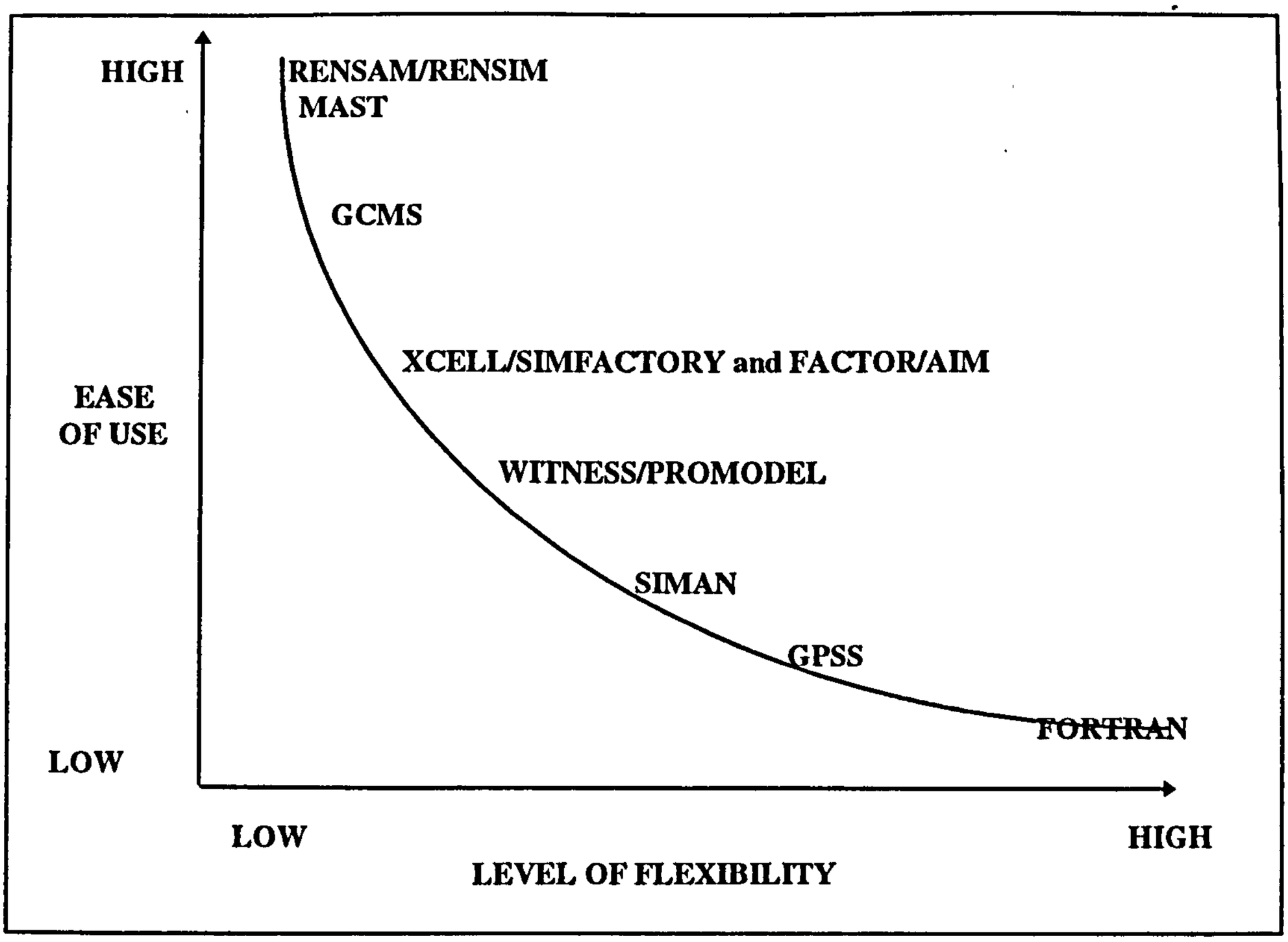


Fig 2.1 The relationship between the level of flexibility and the ease of use of simulation software.

2.4 Development of simulation modelling constructs

Any model represents a view of a system that it tries to depict. Development of a modelling tool similarly has to be based on a generic view of the domain or the world it is designed to depict, and requires modelling constructs or concepts that are designed to express that view. Early work on discrete event simulation was concerned with the development of such basic modelling concepts so as to give it a distinctive methodology and facilitate the creation of a specially designed tool for it. Three primitive constructs were developed- events, activities and processes, which were used as the basis for describing

the dynamic systems that were to be modelled. The three concepts gave rise to very distinctive modelling approaches, or world views as they are commonly called, in the simulation software that subsequently emerged- event scheduling (ES), activity scanning (AS) and process interaction (PI).

2.4.1 Event Scheduling.

In an event based or event scheduling method (Markowitz, Hauser and Karr, 1963) the modeller specifies the real world system in terms of the number of events at which its state or, more precisely, that of one or more of elements changes. Two types of events can occur- consequential events which are dependent on some conditions being satisfied, and scheduled events which are destined to occur at some points in time. Examples of events are arrival of a part at a machine, start of machining operation, machine breakdown, etc. If a simple machining operation in a system consisting of just that one machine is considered, there are three events associated with it- arrival of a part, start machining and end machining. Parts are scheduled to arrive from outside the system at given intervals and, hence, the arrival is a scheduled event but, as a consequence, arrival of the next part is also scheduled and , therefore, it is also a consequential event; another consequential event that may result is the start of machining. The start machining event is dependent on the availability of a part and a free machine, and can result as a consequence of either the arrival of a part or the end of machining. The routines corresponding to the start

machining and end machining events are shown in Fig 2.2 and Fig 2.3 respectively.

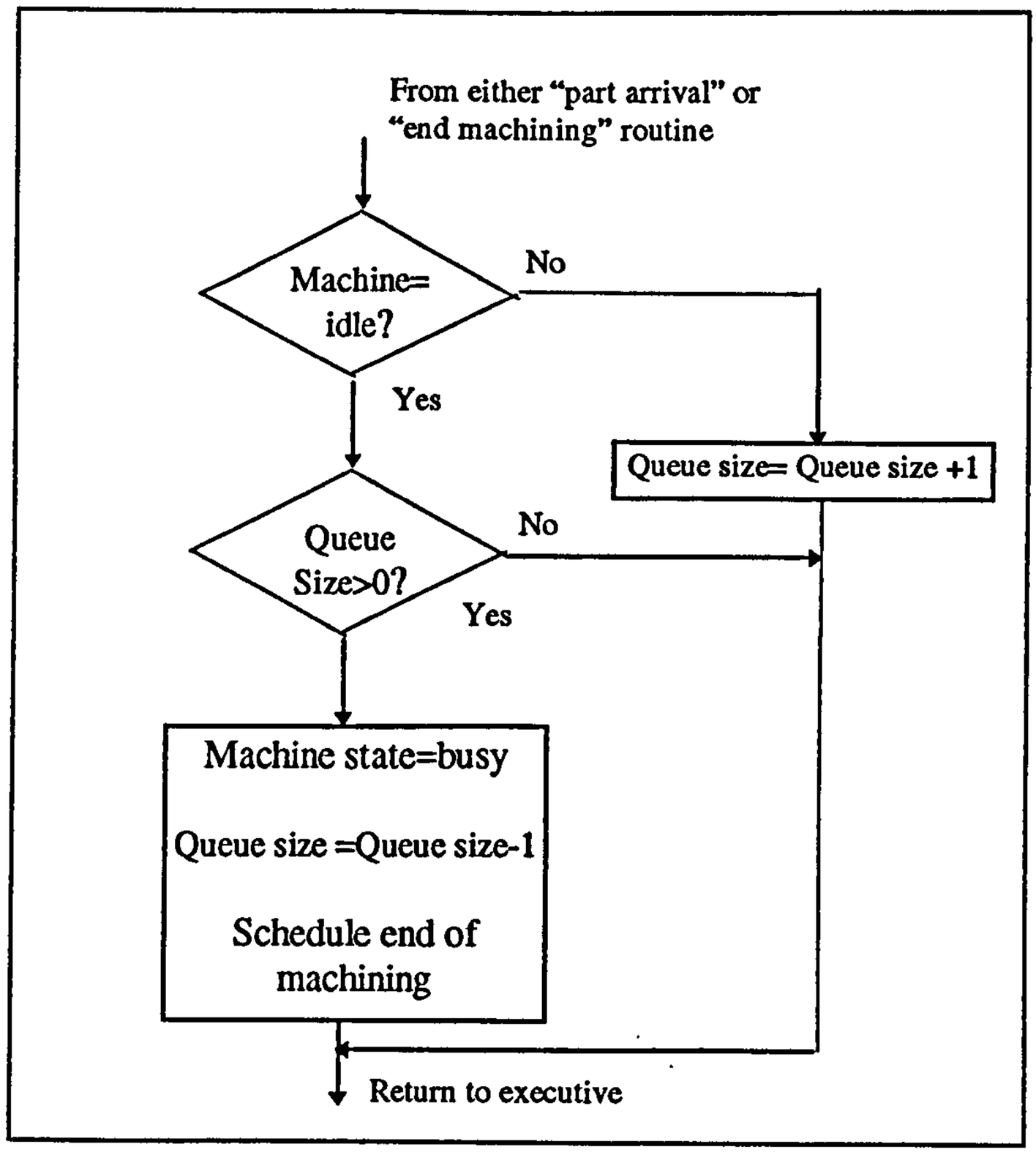


Fig 2.2 Start machining event

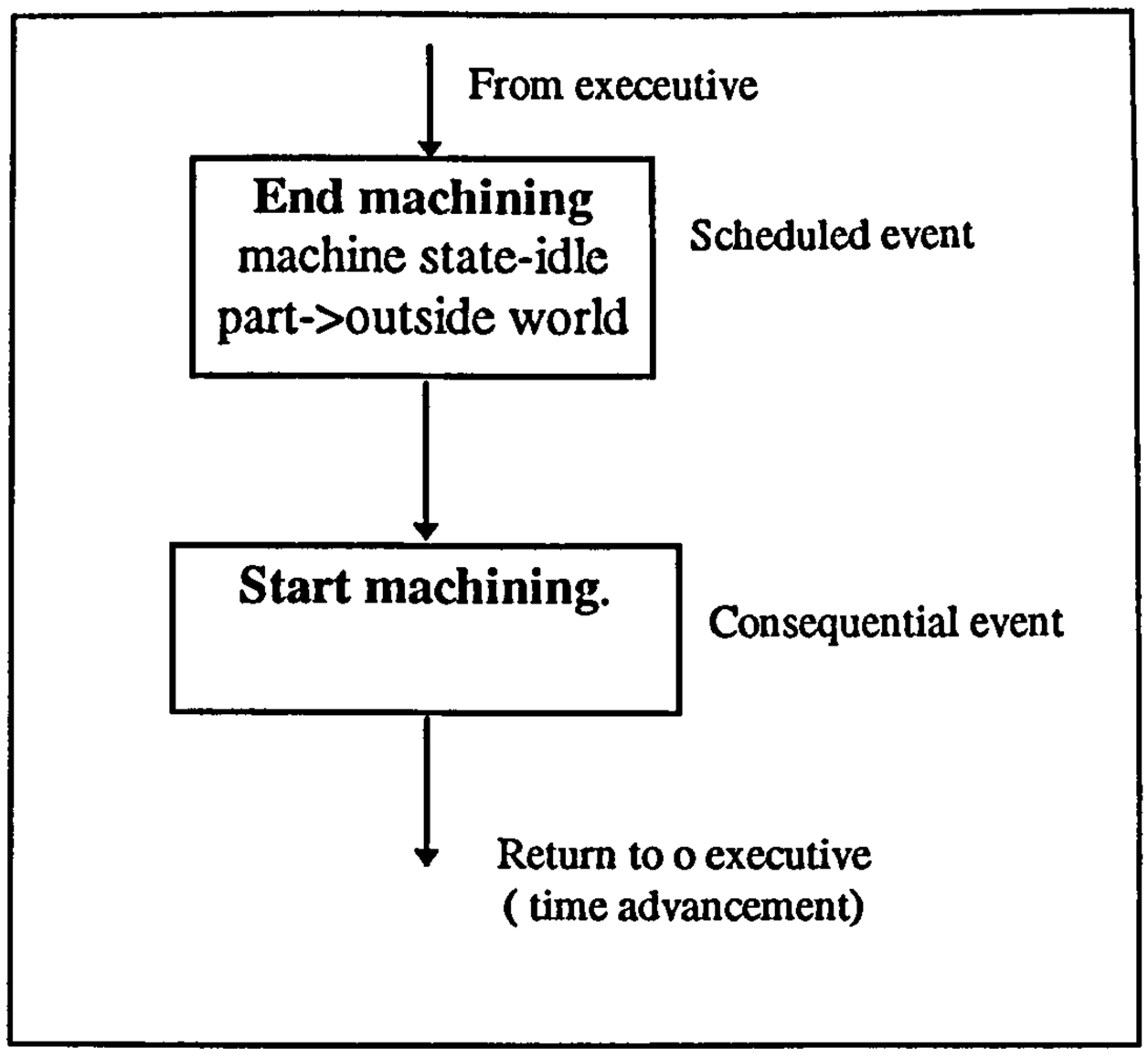


Fig 2.3 End of machining event routine.

An event based executive scans a list of future events, determines the time of the most imminent event and moves the simulation clock forward to this time. It then executes the scheduled event as well as the corresponding user specified consequential events (event phase), before returning to the time phase. The simulation progresses in time by iterating through these two phases. The operation of the executive is illustrated in fig 2.4.

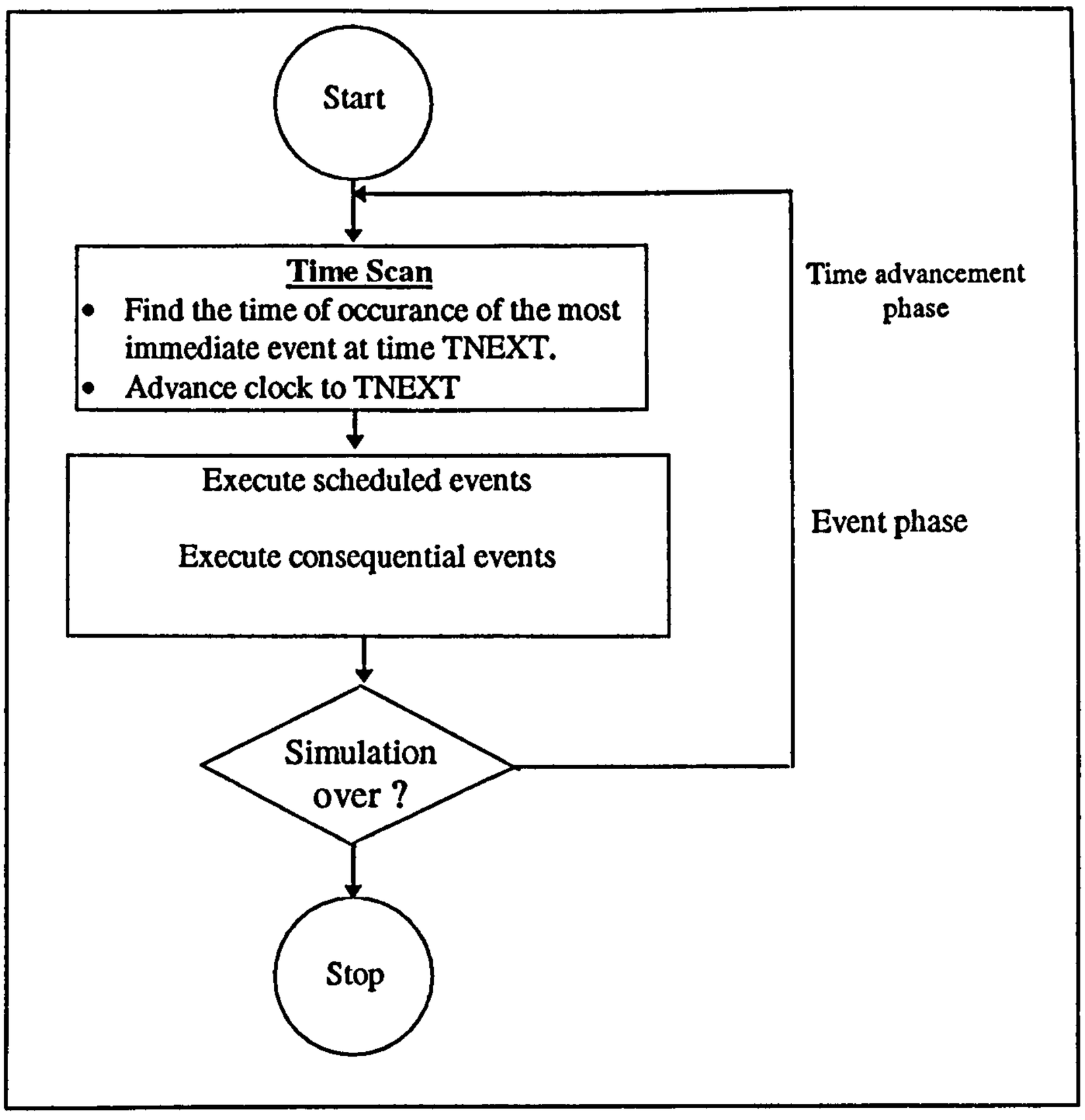


Fig 2.4 The operation of an event based executive

In this approach the user not only specifies the real world system using events as the building block or modelling constructs, but also the consequences that result when a scheduled event takes place. This exact specification of the behaviour of the system makes the ES approach computationally efficient since the executive does not waste any unnecessary time trying to test consequential events that are clearly not affected by the occurrence of a particular scheduled event.

An event is the basic building block in simulation modeling, and internally all simulation software works on the basis of events. However, to raise the level of model specification and provide a degree of modularity in its definition two higher levels of modelling constructs were soon developed-activities (mainly from research in UK) and process (resulting mainly from research in USA).

2.4.2 Activity Scanning.

An activity has a duration determined by the interval between the start and end events. A true activity definition consists of a condition under which it can start, changes in state of the system entities at start, its duration, and the changes in state of system entities when it ends; it effectively combines the start and end events corresponding to the activity. Early attempts at the development of an activity structure (Buxton and Laski, 1962), however, retained the separate definitions of the start and end events, but removed the onus on the user to define the relationship between the scheduled and consequential events. All events, scheduled or consequential, are defined independently, each with its own start condition; the two phase executive automatically tries to start each event after the time phase. The executive is illustrated in Fig 2.5. The sequence in which the events are specified is extremely important as the executive scans them in that order when attempting to start them; an end machining event would need to be specified prior to the corresponding start machining event to ensure that a machine

that becomes idle on completion of an operation is immediately available for the following operation.

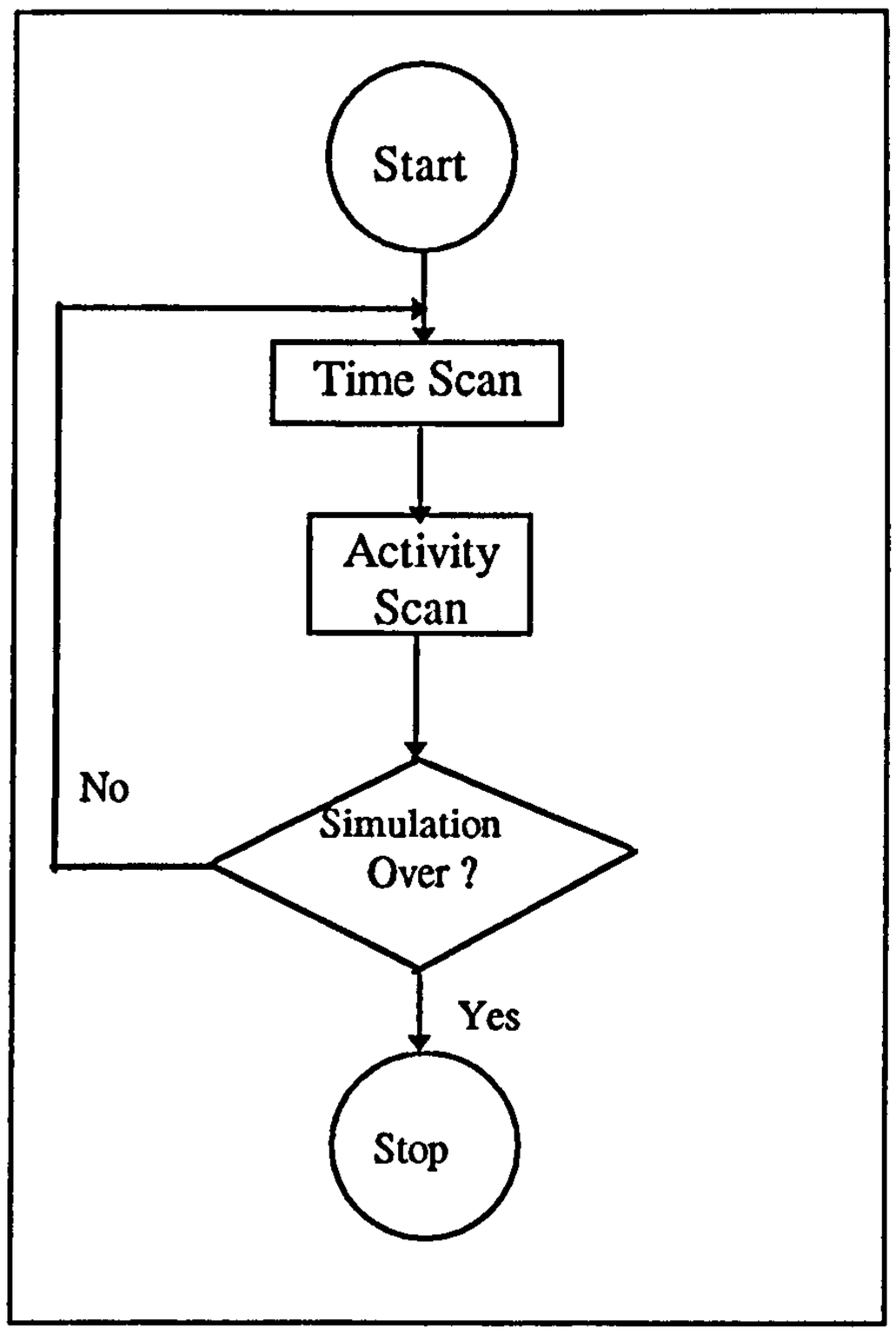


Fig 2.5 Activity based executive

Such a early version of the activity scanning method was soon replaced by the three phase approach (Tocher,1963). There were two developments associated with this latter and true activity based approach, Model specification could be truly modulerised in terms of activities, with the start and end events combined into a single structure; a three phase executive was developed that executes all scheduled events after time advancement, and then proceeds to test each activity in turn to see if it can start

The definition of a simple machining activity is illustrated in Fig 2.6.

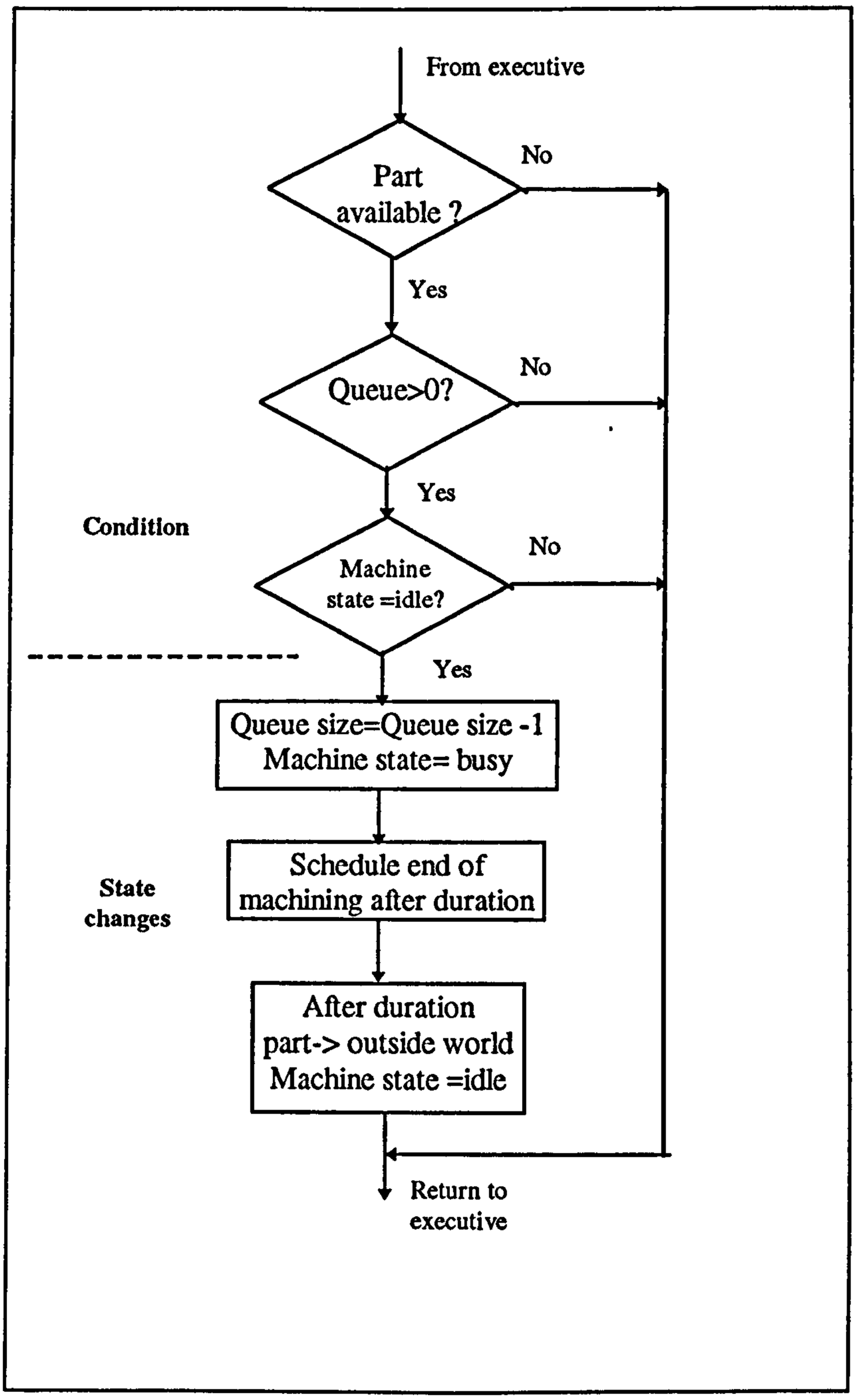


Fig 2.6 Machining activity

The arrival of a part would be modelled in exactly the same way as for event scheduling, but there would be no need to include in its definition any reference to the consequential event of start machining. In the 3 phase approach the events associated with activities are divided into two types. These are:

- ***Bound events*** **(B)** which are unconditional events which should be initiated by the control routines whenever their scheduled time is reached. These are usually the end of activity events. In the machining operation, the state changes associated with the finish machining event will be executed by the simulation executive as soon as the operation has been completed.
- ***Conditional events (C)*** whose execution depends on the either the cooperation of entities and resources or the satisfaction of certain specific condition within the simulation. They are unscheduled events like the start of an activity. The start machining event can be considered a conditional event because it is dependent on status information like whether the machine is available and the queue is empty i.e. the conditions a machine must be free and at least one part must be available have to be met; these are tested during the activity scan phase.

The operation of the executive is shown in Fig 2.7.

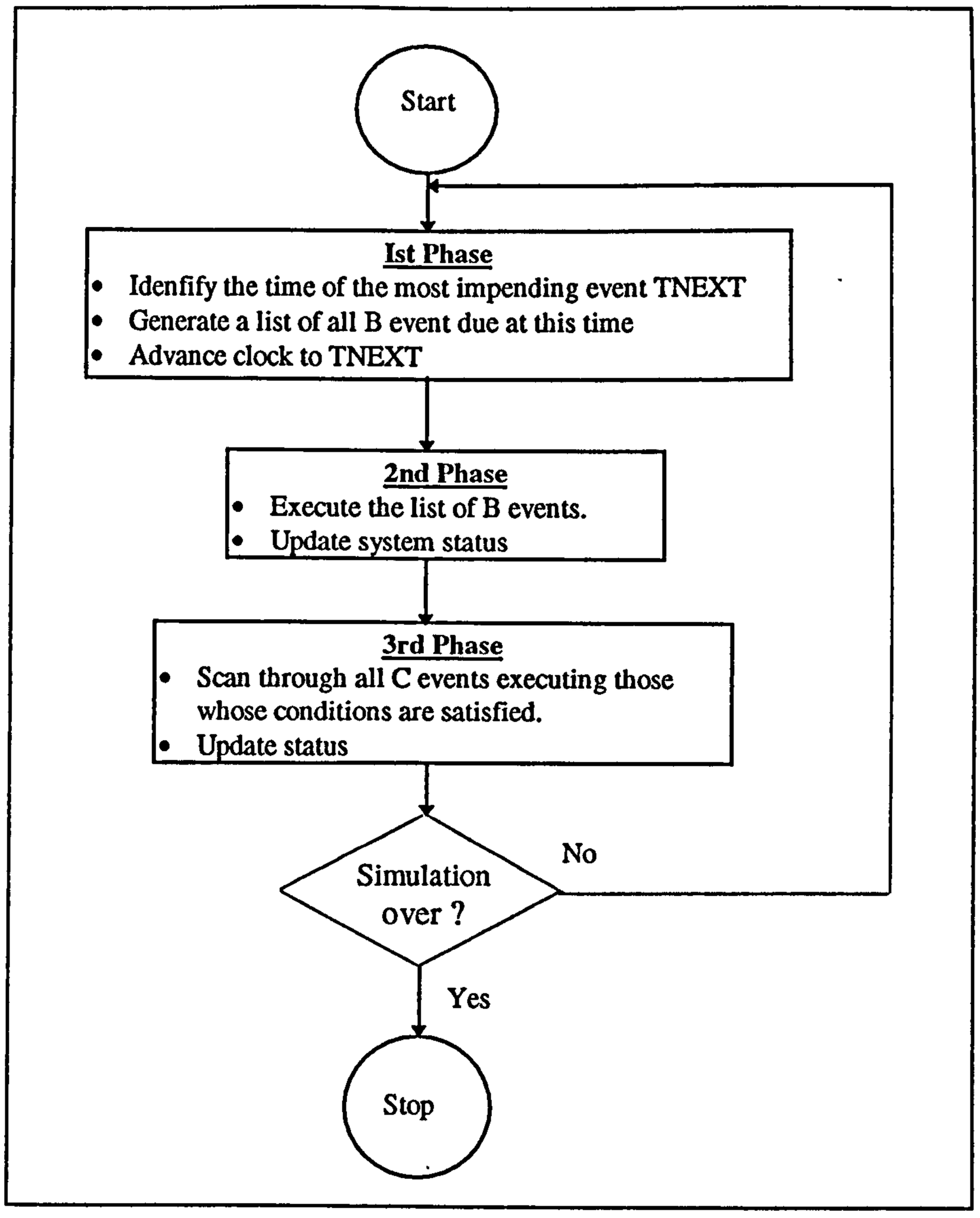


Fig 2.7 The operation of a 3 phase executive.

The (true) activity based approach provides a highly modular structure to the model specification, and removes some of the specification task from the user (i.e. the need to specify consequential events associated with each scheduled event); however, the latter also makes the approach computationally far less efficient than the event scheduling method since the entire activity list has to be scanned.

2.4.3 Process Interaction

In the process interaction (PI) approach (Gordon, 1961) the real system is envisaged as a set of processes, representing the sequence of operations through which temporary[1] entities in the system (i.e. parts) pass during their life cycle within the system. A process can be thought of as routines that encompass the history of the part as it passes through the system. The approach models system from the perspective of the parts as they move through the system, in contrast to the activity and event approaches which are based on the perspective of the resources and the operations they are engaged in. It is the simulation software that translates the processes for each part into a sequence of events.

As an example consider the single machining operation. The part process is then the sequence of operations:-part arrives, waits until it is at the front of the queue, seizes machines, waits until service complete and then leaves machine. The flow for this process is shown in Fig 2.8. It can be seen that there are three points when control is returned to the executive, at points corresponding to when the part is obstructed or delayed. There are two types of delays: ***unconditional*** when a part remains at a certain point in the process until a pre-defined processing time has expired; and ***conditional***

---

[1] Simulation entities are divided into two categories:- permanent and temporary. Permanent entities are those like machines that remain in the system throughout the duration of the simulation. Temporary entities like parts and assemblies are those that pass through the system and are of no interest once they have left.

when the delay is dependent on the state of the system e.g. the part remains in the queue until the machine is available and the part is selected as the next for processing. An attempt is made to reactivate the progress of a part when an unconditional delay time expires, but it may experience a further conditional delay.

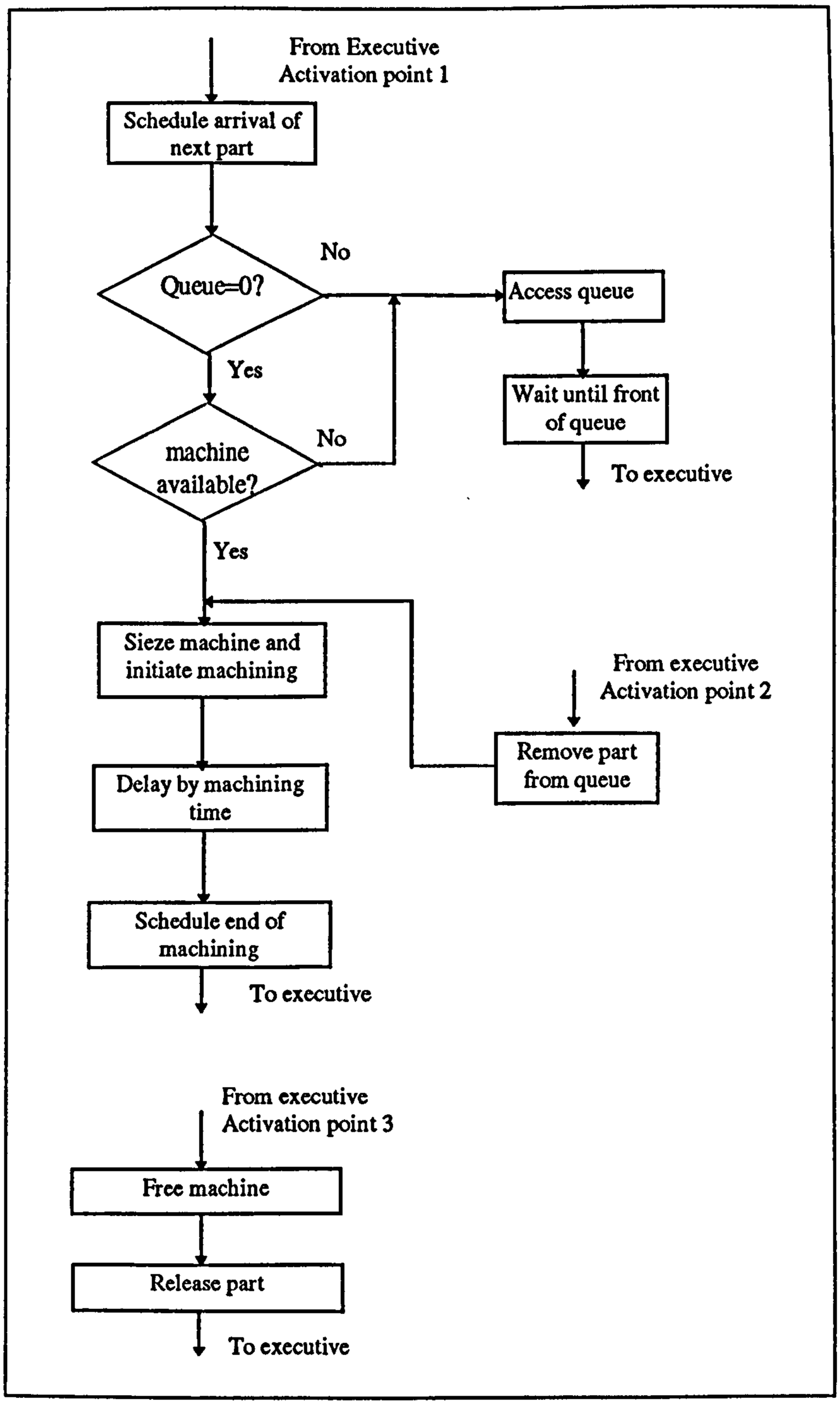


Fig 2.8 A simple machining operation process

A operation of a process interaction executive is shown in Fig 2.9

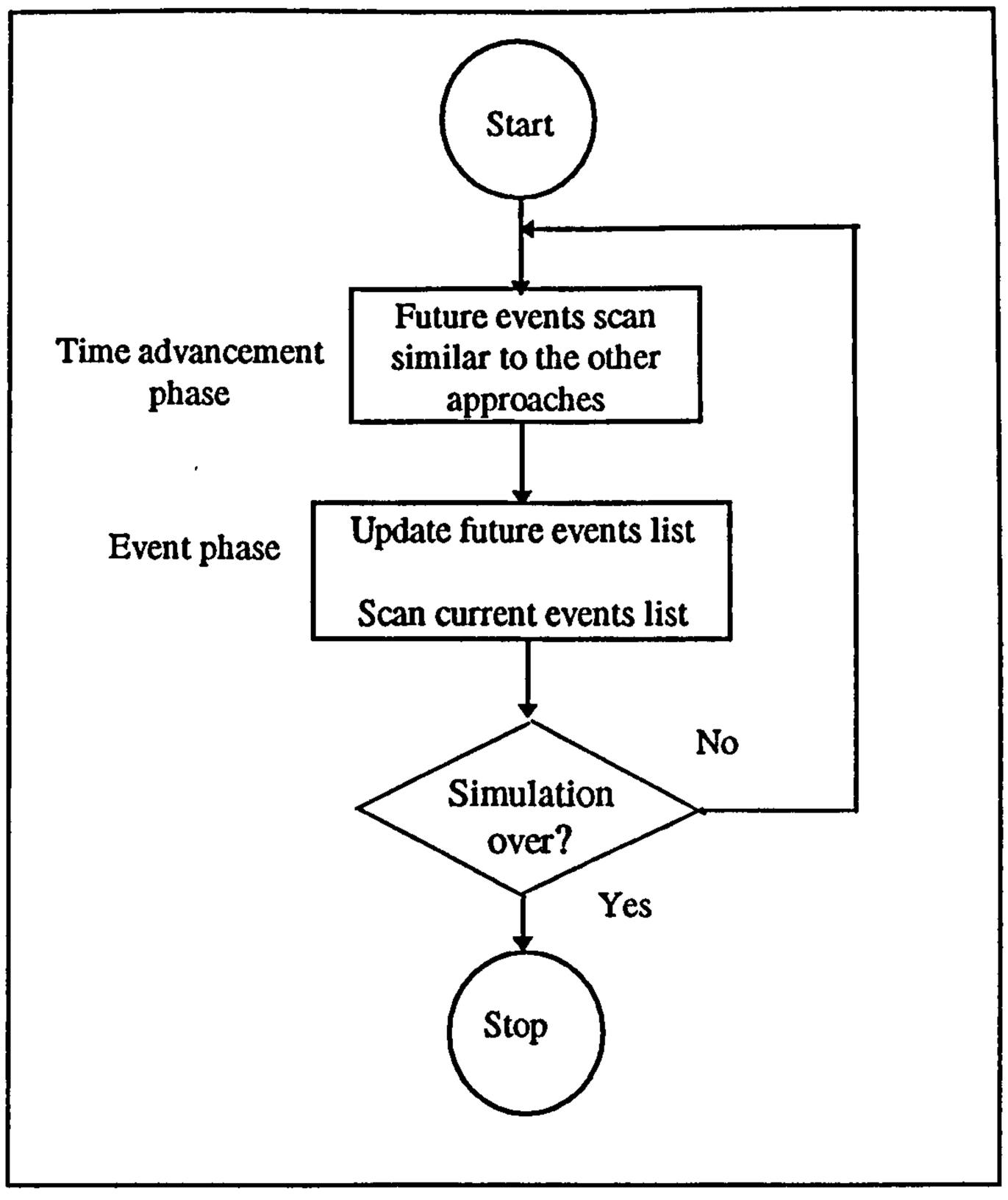


Fig 2.9 Process interaction executive

In practice, most process interaction software takes a more pragmatic approach and in the event phase try to reactivate all resources (permanent entities) in the system that are currently idle. This is almost always computationally more efficient since there are usually a lesser number of resources than parts. In either case, a two phase executive is used and, hence, the PI approach is computationally more efficient than the activity based approach; it is, however, still computationally less efficient than the event based approach since the user does not guide the search process as he does in the case of the latter.

2.4.4 Advantages and disadvantages of approaches

From a programming point of view the activity based approach is the most efficient because the model is composed of small activity blocks making it easier to write, debug, modify and enhance. It is also stated by Pidd (1988) that this approach "greatly eases the process of 'top-down' design favored in structured programming". Therefore, from the viewpoint of ease of model specification it is superior to the other two approaches. However it is very inefficient in its use of computer time because a significant amount of computer time is wasted in scanning the complete set of activity routines each time the simulation clock is advanced.

The process interaction approach is more in tune with a program structure that is material flow oriented; hence, it has advantages in modelling manufacturing systems with a large number and variety of parts, and general purpose resources e.g. as in jobshops. An added advantage of the approach is that it is synonymous with the building block approach compatible with the simulation developer's notion of the life history of a part as it moves through a system. However, the approach is less modular than the activity based method, processes are more difficult to code, and it is difficult to model specific constraints of resources (other than general availability). Also, the executive is more complex to implement. For a more thorough explanation of the advantages and disadvantages of the three approaches the reader should consult Pidd (1988) and Fishman (1978).

An example of a 3-phase activity based simulation language is Hocus, whereas examples of process and event based languages are SIMAN and See-Why respectively. Table 2.1 below contains some languages and their world view orientation.

World View	Language/Package
Events	FORSS (IGHT) OPTIK GASP II. IV SEE-WHY SIMSCSRIPT II.5[2]
3-Phase Activity	ECSL HOCUS SIMON
Process	GPSS Q-GERT SIMAN/CINEMA SIMULA[3] SLAM II SIMSCRIPT II.5

TABLE 2.1 Simulation languages and their world views

2.5 Historical perspective on the development of simulation software

Ease of model specification depends not only on the modelling structure that is available to the user, but also on the software environment in which the particular simulation system is implemented. Simulation software has developed considerably from the early days of models written in 3rd generation general purpose computer languages to the present day generic

---

[2] As reported in Law and Kelton(1982) SIMSCRIPT II.5 can be used in both process interaction and event scheduling modes.

[3] Although SIMULA is an object oriented language, its world view, as reported by Pidd(1990) and Nygaard(1978), is process interaction.

simulators; many features have been added over the years to aid ease of use, productivity, validation and presentation.

A visual depiction of how simulation software has progressed from the 1950's is given in Fig 2.10, whilst Table 2.2 shows how simulation software for manufacturing has evolved from 3rd Generation General Purpose Computer Languages to Generic Simulators.

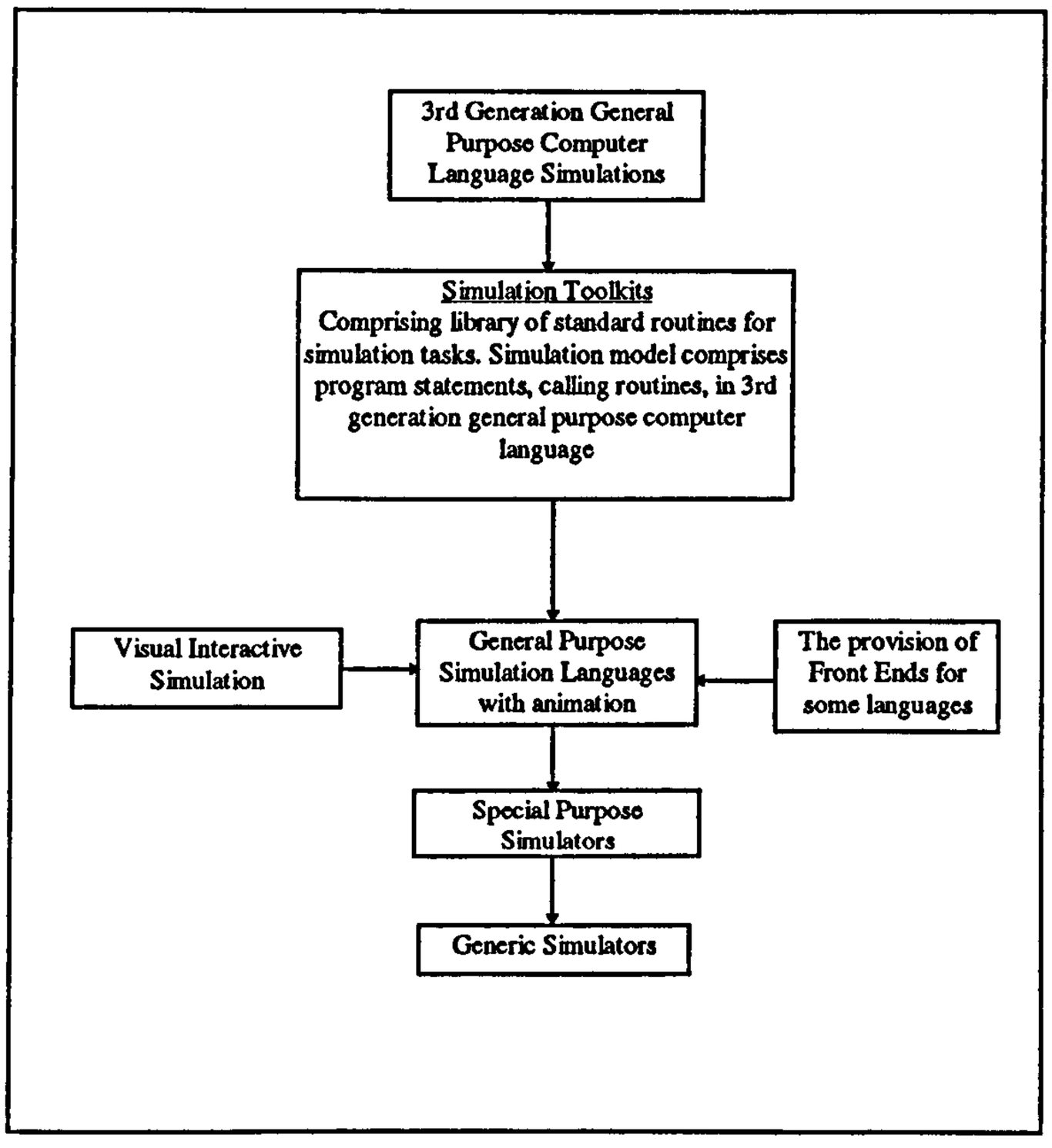


Fig 2.10 The development of simulation

Year		3rd Generation General Purpose Computer Languages	Simulation Toolkits	General Purpose Simulation Languages	[4]Simulators
Up to	1960	Fortran			
	1961			GPSS	
	1963		SIMSCRIPT	CSL	
	1964		GASP		
	1966			SIMULA	
About	1970				
	1972			Q-GERT	
	1976			HOCUS	
	1977			GPSS/H	
	1979			SLAM II	
	1980		SEE-WHY	ECSL	MAST
	1983			SIMAN	
				SIMSCRIPT II.5	
	1984		DISC		AutoMod
Since	1985		CSIM		WITNESS
					XCELL+
					SIMFACTORY
					ProModelPC

Table 2.2 The Development of Simulation Software

[4] This applies to both special purpose simulators like MAST and generic simulators like WITNESS

2.5.1 Use of 3rd generation general purpose computer languages

Up until about 1960, before the advent of simulation languages, models were developed in general purpose computer languages like FORTRAN. It involved writing routines for facilities which would later be available in simulation languages, e.g. for the simulation executive, random number generator, sampling from probability distributions, statistics collection, etc. The user would then construct a simulation model by writing program segments to describe the logic of the modelled system calling the routines that performed the standard functions.

The advantages of using 3rd generation general purpose computer languages are:

1. Modellers usually know a general purpose computer language.
2. Portability, since most general purpose computer languages can be run on most computer operating systems.
3. Greater flexibility.

The disadvantages:

1. The writing of all the general simulation facilities is time consuming, increasing model development time.
2. The entire program has to debugged with no error detection for tracking the progress of entities through the simulation. An added hindrance is that the

error messages are in the host language, and not specific to simulation modelling.

2.5.2 Toolkits

Toolkits resulted from the realisation that certain features are required in any DES, and that a substantial amount of the code in the early simulations models, written in 3rd generation computer languages was reusable. These toolkits provide libraries of routines for time handling, moving entities, random number generation, statistics collection, etc. The simulation model is then developed by writing a 3rd generation computer program that calls these routines.

The first toolkit was SIMSCRIPT (Markowitz, Hauser and Karr, 1963) which provided FORTRAN routines for use in DES. To develop the model the user writes a FORTRAN program to call these routines. Other systems which adopted the same approach were GASP (Kiviat and Colker, 1964), SIMON (Hills, 1965), SEE-WHY (Fiddy, *et al*, 1981). SIMON was originally a collection of ALGOL routines, while later toolkits like CSIM (Schwetman,1988), SMPL (MacDougall,1987) and DISC (Selvaraj; Blair; etal,1990) provided routines written in C.

2.5.3 General purpose simulation languages

These are programming languages which have their own vocabulary and syntax oriented towards the specifics of simulation programming, and are often termed Statement Description Languages (SDL). GPSS (Gordon, 1961) is probably the earliest simulation language and was developed in the USA by IBM. It is a process interaction simulation language in which entities flow through blocks in a block diagram, and it culminated in the version GPSS/H in 1983. SIMAN (Pegden, 1985) another process interaction language, was the first to separate the model and the data to run the model. It also allowed event based modelling via the incorporation of an event statement in the model frame to call user written FORTRAN routines to perform computations not possible using the standard process interaction approach; in addition, it provided a block diagram interface as a possible method for model specification.

Most of the early work in the UK on the development of simulation languages was performed by Tocher in the steel industry, resulting in one of the first simulation packages GSP (General Simulation Package) (Tocher, 1963). The language that evolved from this research effort was CSL (Buxton and Laski, 1962) which was further developed by Clementson into ECSL (Clementson, 1982).

The research paths followed by the general purpose languages mentioned thus far have been solely concerned with SDL. However, SIMSCRIPT started off as

a toolkit and eventually became a statement description simulation language culminating in SIMSCRIPT II.5 (Russel,1983). GASP also followed this development path and evolved into GASP IV (Pritsker,1974), and was later combined with Q-GERT (Pritsker, 1972) to form SLAM (Pritsker and Pegden, 1979). It should be noted, as stated in Law and Kelton (1982), that SIMSCRIPT II.5 and SLAM can be used in both event and process orientations.

The features usually provided by these general purpose computer languages are:

1. Time handling. This maintains the simulation clock, the list of future events and handles event logic.
2. Data structures to hold information about simulation entities, and for taking entities into and out of queues.
3. Initialisation facilities to set conditions in the model when the run begins.
4. Random number generation.
5. Distribution sampling e.g. for the duration of activities and the processes of the real world which are often stochastic.
6. Error checks and diagnostics.
7. Collection of statistics..
8. Report generation in the form of numerical printouts, histograms, time series and pie charts.

The advantages of simulation languages are:

1. Powerful software tailored to the purpose of simulation with the provision of the majority of common facilities required for simulation programming, resulting in a decrease in model development time.
2. Need for fewer lines of user written code.
3. The models are easier to change.
4. The provision of error detection to check for illogical or impossible situations with the provision of suitable domain (simulation) specific error messages.

However specialist training is required in the use of the simulation language.

2.5.4 Object Oriented Simulation

Object oriented simulation focuses on the objects that make up the system as opposed to the overall function of the system. These objects are encapsulated so that they can hide the data, and procedures, which define their behaviour. Objects interact by passing messages which usually result in the execution of certain actions to change system status. A machine can be thought of as an object which has a set of attributes that define its state and a set of operations for manipulating this state. As an example, an attribute of a machine may define its type i.e. whether it is a single, or production, or assembly machine, etc, and the operations relate to the start of processing, its setup, repair, etc. An important characteristic of object oriented simulation is that objects may be grouped into classes; for example there may be a class of machines related to milling operations. It is extremely useful for simulation software reuse

because developers can use objects that meet their modelling requirements from existing classes.

A particularly useful aspect of object oriented programming is that of inheritance which allows a new class to be defined as a refinement of an existing class. The new class inherits all the attributes and operations of the existing class and can have new attributes and operations to distinguish it, and any changes made to the attributes and operations of the existing class are passed to the new class. This can be viewed as an inheritance tree of objects, called the class hierarchy, which describes the hierarchy of objects which have been derived from one another.

In an object oriented simulation some of the objects can be thought of as active because they execute independently and concurrently with other active objects. These active objects schedule events for each other to define when state changes can occur by the passing of messages between themselves.

There is a wide variety of object oriented languages with some providing facilities geared towards simulation model building. The first object oriented simulation language was SIMULA (Dahl and Nygaard, 1966) which was developed from the early 60's onwards at the Norwegian Computing Centre, and makes use of the process interaction modelling approach. It uses objects to describe all entities, and allows the objects to be grouped in classes. Each object has a set of attributes and a process, which is a sequence of activities in

which the objects of the class engage. The more modern object oriented simulation systems have followed a similar development path as conventional simulation, with facilities added to ease model specification. An example of such a system is SIMPLE++ (AESOP, 1994) which provides a library of primitive objects that can be combined with user written models to build new models.

2.5.5 Simulators

These are generalised models of a specific or generic type of system developed in such a manner that the modification of certain parameters can be used to achieve different models. There are two types: special purpose which can model certain types of very specific systems e.g. FMS, automated warehouses, AGV systems, etc; generic which apply to a much wider domain, with most developed for manufacturing applications.

In the beginning the motivation behind simulators was to ease model specification by use of improved user interfaces, and by eliminating all programming. The specification method was further enhanced by the use of modelling primitives or modules which attempt to imitate the characteristics of real world objects (resources, materials, conveyors, etc.) However, as systems developed it was evident that some form of programming had to be retained in an effort to preserve some modelling flexibility.

2.5.5.1 Special Purpose Simulators

These systems require no programming and are effectively very large models specific to a certain type of manufacturing system. Although these systems don't require any simulation programming knowledge, they do require the user to learn the input specification and, therefore, require the user to possess some simulation expertise. The ease of use of these systems is usually at the expense of their flexibility, and they are usually of little use outside their application domain.

The MAST system (Lenz, 1989), a development of GCMS (Generalised Computerised Manufacturing Simulator (Lenz, and Talavage, 1977), was the first simulator, and is an integrated environment particularly suited to the modelling of FMS. The model requires data specific to parts, workstations, conveyors and/or AGVs, in-process storage and system layout. In addition it provides a library of scheduling rules for parts, machine selection, AGV routing, etc. The model is completely data-driven and no programming is required, and the layout is defined by points and their links. Other systems specific to FMS are RENSAM/RENVIS (O'Keefe, and Haddock,1991) and MAP/1 (Rolston, 1985), whilst PROPHET (ICI, 1990) is a simulator used in certain applications in the chemical industry.

The advantages and disadvantages (O'Keefe and Haddock, 1991) of data-driven models can be summarised as:

Advantages

1. If the input data is valid, then the statistics that come out will be valid. This is because the tools have been thoroughly tested. In traditional languages if the programmer has incorrectly coded the model, the

statistics may be correct in their values but not in their interpretation; in data-driven systems the model is more reliable since experimentation is achieved by altering the data set, not by altering generic code specifying the model..

2. Ease of use since no programming is required.
3. Rapid model development due to lack of iterative development, refinement and debugging.
4. Proper statistics collection and analysis are built in and not left to the user.

Disadvantages

1. Different organisations have different terminology and view systems in different ways, and this may limit the use of a particular simulator. For example, RENSAM uses the concept of fixtures rather than pallets and sees the number of parts in FMS being constrained by the number of fixtures rather than the number of pallets. Some organisations in contrast may view pallets as more critical and find the Modular FMS simulator (Montazeri, Gelders, and Van Wassenhove, 1988), written in FORTRAN and GPSS, more amenable, since it uses pallets rather than fixtures
2. These systems often require more data than is required or available, resulting in over specification. This is because the actual model may contain low level programming statements that require the satisfaction of certain data requirements. In RENSAM, for example, Materials Handling System (MHS) movement times are specified as a matrix of times between stations; for a six station system a 6*6 matrix would be required; if, for example, the MHS is underutilised, we may wish to represent all travel times between all stations as a single constant for simplicity, we

would still have to specify the time in the matrix format to an unnecessary level of detail.

3. Incorrect input data can have the following consequences:
   a. If the simulator is robust, the problem would not be detected until the simulation is executed and it may not even then be detected if the simulator was really robust.
   b. If it is fragile, it will produce error messages related to the underlying simulation program and not the data input, leaving the user to find the relationship between the error message and the data.
4. Ease of use is achieved at the cost of reduced machine execution efficiency.
5. They will always reach the limits of their capability.

It was highlighted by Bevans (1982) that a truly user friendly FMS simulator would ask questions of the user concerning the FMS to be simulated, and after performing the simulation would immediately display the results[5]. He indicated, however, that they fall well short of this level of user friendliness and can only be used by computer knowledgeable persons.

2.5.5.2 Generic Simulators

In an attempt to strike a balance between ease of use and flexibility; a new generation of simulation software have now come on the market since the mid 1980's referred to as generic simulators, they have been primarily targeted at general manufacturing system applications and, hence, are also referred to as

[5] However it should be noted that they may not be displayed in the terms the user needs.

manufacturing simulators. The development of these systems attempted to combine the ease of use of special purpose simulators with the flexibility of the general purpose simulation languages. They do not require the user to write an extensive simulation program but supply the input specification for a specific domain, that is broader than in the case of special purpose simulators, and in most cases encompasses the entire manufacturing domain. The input specification is in the form of a combination of data and high level language using special modelling constructs of the particular system. They are considerably easier to use requiring less programming expertise than if general purpose simulation languages were used. However they do sacrifice a degree of modelling flexibility in the process.

In an attempt to limit this reduced flexibility some of the systems like WITNESS and PROMODEL have in-built programming languages which give them greater flexibility than completely data-driven models. For modelling complex features, the user written program may not always remain simple; however, since most of the data is contained in a data driven format, the programming effort required is limited, usually needed for expressing complex logic.

The manufacturing simulators both raised the level of model specification and provided a high degree of modularity in it by developing modelling constructs based on real world objects, e.g. machines, parts, process routes, labour, etc.

Model specification was also made easier by parallel developments in user interface techniques, e.g. GUI, pop-up menus, dialogue systems. The greater ease with which models can be developed has made these tools accessible to a much wider user base than previous simulation systems, and explains their dominance in the market. The subject of generic manufacturing simulators is discussed in greater detail in chapter 4.

Developments in the field of computer graphics has also contributed to important advances in simulation software, and these are briefly explained in the next section.

2.6 Advent of animation and visual interactive simulation

An important breakthrough in simulation software was the advent of animation. Animation is a method which utilises computer graphics to produce a visual display of the temporary entities moving through the system comprising of permanent entities. Typically the displays are in colour allowing colour schemes for different entity status and icons for representing different types of permanent entities. The benefits of animation are:

1. It can be used to aid the production and debugging of simulation models. Animation of the mechanism and behaviour of the simulation model allows the detection of logical errors introduced during model development.
2. Improves communication and presentation by providing a decision maker with an insight into the model and an explanation of the solution.

3. Bottleneck Analysis. Bottlenecks are easily detected because the animation allows the observation of several simultaneous and inter-related events. It also allows an insight into the preconditions that result in the bottleneck.
4. Provides an insight into whether the assumptions inherent in the model have resulted in a valid representation of the modelled system.

It should however be mentioned that despite its advantages animation is no substitute for rigorous model validation and statistical output analysis.

Many of the early animation systems, like FORS to form FORSIGHT, and SIMON to form SIMON/G, were added to existing general purpose simulations languages as post-processors, where the animation is played after the completion of the simulation run. Some systems like, HOCUS and CINEMA are examples of concurrent animation systems where they run concurrently with the simulation; the display is updated as each event occurs and work pieces move from machine to machine, or as the state of resource changes. However, in most such systems , the animation and the simulation models are developed separately.

An important development in simulation systems in the early 1980's, and an aid to generic simulators, was the advent of SEEWHY (Fiddy, *et al*, 1981) a visual interactive simulation system which allowed the simultaneous creation of the simulation model and animation. It uses a library of FORTRAN routines which are designed to ease the process of building visual interactive simulation models. In addition to the usual simulation routines, it provides

convenient ways of creating and moving graphic images on-screen. This form of Visual Interactive Simulation was made possible by advances in graphics and animation, and its advent was deemed the most important advance in simulation since the development of general purpose simulation languages. This allowed models to be created gradually by allowing the user to stop model execution to correct modelling errors, which may have become evident visually.

These systems provided a means of fine tuning the simulation without having to perform multiple full length simulation runs, and graphical output such as histograms can be constanly updated as computations proceed, increasing the user's understanding of the simulation results and, therefore, their credibility. It helps the user to visualise how models are put together, how the parts interact and how to modify the model structure.

In See-Why, for example, the following categories (Carrie, 1988) of interaction are provided: run control; display changes and output; element inspection and manipulation; saving and restoring the model; user's own interactions. The concept of visual interactive simulation has now been incorporated to a greater or lesser extent in most modern simulation systems.

2.7 Discussion

From the very early days of simulation software there has been a constant drive towards easing the task of model specification. This was initially attempted in one of two ways: via the world view or the structural approach (event scheduling, 3-phase, and process interaction) used to conceptualise the modelled system; and via the software or environmental approach (simulation toolkits, and general purpose simulation languages) used for implementing the model.

There has also been a third approach to aid model specification, through research into automatic model generation (discussed in detail in chapter 3), which attempted to reduce the modelling burden by incorporating domain, simulation and target language knowledge in the tool; more recent work has been concerned with various Artificial Intelligence (AI) tools and concepts in simulation software. In parallel to such work, and with similar motivation, manufacturing simulators came into existence which, while not using AI knowledge representation techniques, do use AI concepts of knowledge representation and real world objects (modelling primitives which map real world entities), for representing manufacturing domain knowledge. The result of this in-built knowledge has been that simulation models can now be developed by people with little computer expertise.

These developments have resulted in the research trend moving away from general purpose simulation languages towards simulators. The task of model specification was further aided by advances in user interfaces; this was achieved by providing WINDOWS based environments comprising iconic pallets for animation creation and dialogues for model data entry. Also, work on the development of generic simulators aimed to extend the applicability to the entire manufacturing domain while still preserving a degree of modelling flexibility by providing an underlying programming language within a data driven framework. This made it possible to develop more complex models with less simulation training and in less time than was typically the case with 3rd generation computer programming and general purpose simulation languages. The flexibility of these systems, to express logic not afforded within the main modelling environment, is further enhanced by most systems providing the modellers with the option of dropping out into computer languages like FORTRAN and C.

Although the development of general purpose simulation languages may have stagnated, their employment is not entirely redundant. Sometimes the real world system may be too complex to be easily represented in a generic simulator, or its form may be incompatible with the underlying modelling structure provided by the latter; it may be either impossible or too difficult to represent the system in a satisfactory and easy to understand manner; in such cases it may be advisable to use a general purpose simulation language or toolkit.

CHAPTER 3

SIMULATION CODE GENERATORS, AUTOMATIC MODELLING SYSTEMS AND ARTIFICIAL INTELLIGENCE BASED SIMULATION SYSTEMS

3.1 Introduction

Much of the research in the area of Discrete Event Simulation from the early days has been geared towards making the model specification process easier. Two approaches for this were discussed in the previous chapter:-of the world views to provide consistent modelling structures and greater modularity in model specification (structure) and by improving the simulation software (environment).

A third approach evident in the research efforts was the development of automatic simulation programming systems. The objective of such systems was to take a specification of a system from the designer (where informality is desirable) and utilising automated programming techniques to construct the model without any further user intervention. They have gone some way in removing the modelling burden from the user by attempting to store simulation and/or modelling expertise within a computer program.

Barstow (1983) states that "An automatic programming system allows a computationally naive user to describe problems using the natural items and concepts of a domain with informality, imprecision and omission of detail. An automatic programming system produces programs that run on real data to effect useful computations and are reliable and efficient enough for routine

use." These systems effectively restyled the way the programs are specified by raising the specification level to a higher more natural level.

The earliest automatic programming systems were the ***code generators***, which were developed prior to the introduction of formal Artificial Intelligence (AI) concepts and tools. However, as AI developed and became prevalent within the general field of automatic programming, so it started to influence automatic simulation programming resulting in the development of ***automatic modelling systems.***

3.2 General Automatic Programming

The aim of automatic simulation programming, which as a subject preceded its use in simulation, was to raise the level of programming to a higher more natural level, in the process removing from the user a substantial amount of the programming burden. The main approaches attempted to continually refine high-level specifications until a low-level implementation in a traditional programming language (Fortran, Pascal, etc) could be automatically generated from them. The aim was to improve the programming environment, so that programs could be constructed more easily and accurately than could be done by a human programmer.

It is in effect the automation of some part of the programming task. The early automatic programming systems were called compilers because previous to this programs were written in machine code. The first system of this type was the first FORTRAN compiler (Backus and Herrick, 1954), which allowed programs to be written in what was then regarded as a high-level language. The development of these high-level languages had a dramatic effect because they allowed representations in a concise and more understandable manner,

with the compiler making a substantial amount of low-level programming decisions. However, since most programs these days are written in similar high-level languages, automatic programming is concerned with a more enhanced program development environment.

The main aim of automatic programming systems is to restyle the way a programmer specifies the program. This is achieved by the specification of the program at a higher more natural level alleviating the more humdrum tasks associated with programming, particularly the requirement for keeping track of large amounts of detail. An additional aim is to generate efficient programs, and involves selecting the best amongst a number of possible implementations.

The general structure of an automatic programming system is shown in Fig 3.1. The main elements are the requirements specification language which is used to enter the program specification, via a user interface, and is converted into a machine understandable representation or data structure. This data structure then acts as data that is transformed by a translator into a lower-level implementation in the target language, which is usually a traditional computer language.

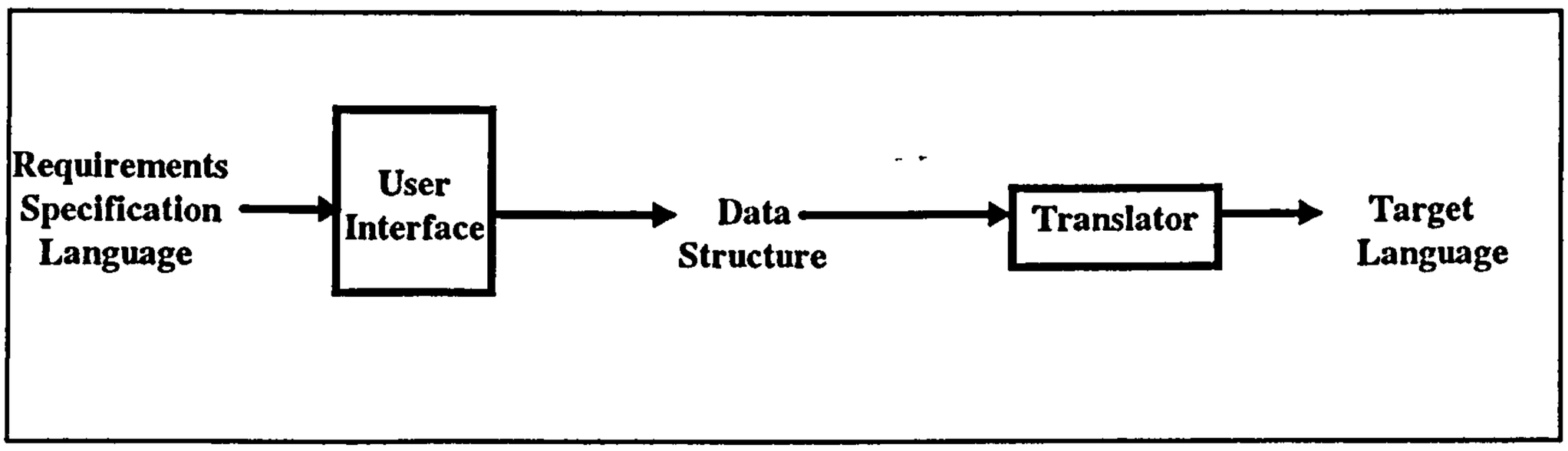


Fig 3.1 The structure of a Automatic Programming System

The requirements specifications languages used in the overall field of automatic programming are natural language, dialogue systems, graphics, examples, logical formalisms and very high level languages. It will become evident from the following sections that all the different types of specification languages are pertinent to simulation program generators and automatic modelling systems, except examples and logical formalisms. It is worth mentioning that model specifications are approximations of a real system, since there is no limit to the amount of detail that can be included in a model. In addition modelling in general is an iterative process which requires continual dialogue between the domain expert and the programmer to determine the user's intent. This will result in changing the specification until the exact requirement is determined. The specifications will eventually become programs carefully crafted, debugged and maintained.

The translation methods for transforming the program specification into a lower level implementation in a traditional target language include procedural, deductive, transformational, and knowledge based methods. In the field of simulation code generators and automatic modelling systems only the procedural and knowledge based methods apply.

The earliest efforts at automatic programming were impeded because of the lack of availability of user friendly program specification methods. Also, users weren't always aware of what they wanted and, even if they were, the specification languages were considered too cumbersome and complicated. This had the adverse effect of some specifications rivalling the generated programs in size, and were often more difficult and error prone to alter.

3.3 Code Generators

3.3.1 General Approach

Code generators were the earliest automatic simulation programming systems for raising the model specification to a higher level of abstraction, and used knowledge representation and translation methods that pre-dated modern AI based approaches. A formal definition (Mathewson, 1984) of a simulation code generator was provided as " A program generator is an interactive software tool that translates the logic of a model described in a relatively general formalism into the code of a simulation language. The facility gives the user the benefits of a simple symbolic input combined with the power of a high level language in which to develop complex models". These early attempts at simplification of the model specification process did not utilise AI, and the systems were written in high level languages like FORTRAN and Pascal. They are however important because they represent the first systems to provide automatic simulation program generation capabilities. They, however, generate only part of the code, requiring subsequent editing by the user for advanced modelling, and in effect only possess knowledge of the target simulation language. In general they require no knowledge of the structure, syntax or semantics of the simulation language[6]. These systems improved modelling times, but still required substantial simulation expertise and experience. In effect there are two types of users (O'Keefe and Haddock, 1991) of simulation code generators:

---

[6] Work has also been done in automating the production of this formal specification(Paul and Doudis, 1986), where a system containing knowledge about the development of Activity Cycle Diagram(ACD) has been developed. The initial system was dialogue based with a subsequent system employing a natural language interface so that a description is compatible with the clients way of thinking. This addition was in effect an attempt to convert the code generator into an automatic programming system.

1. An experienced simulation developer who uses them for fast prototyping, and then edits and further develops.
2. A simulation novice who uses only the generator for relatively simple models.

The main components of such a system are shown in Fig 3.2.

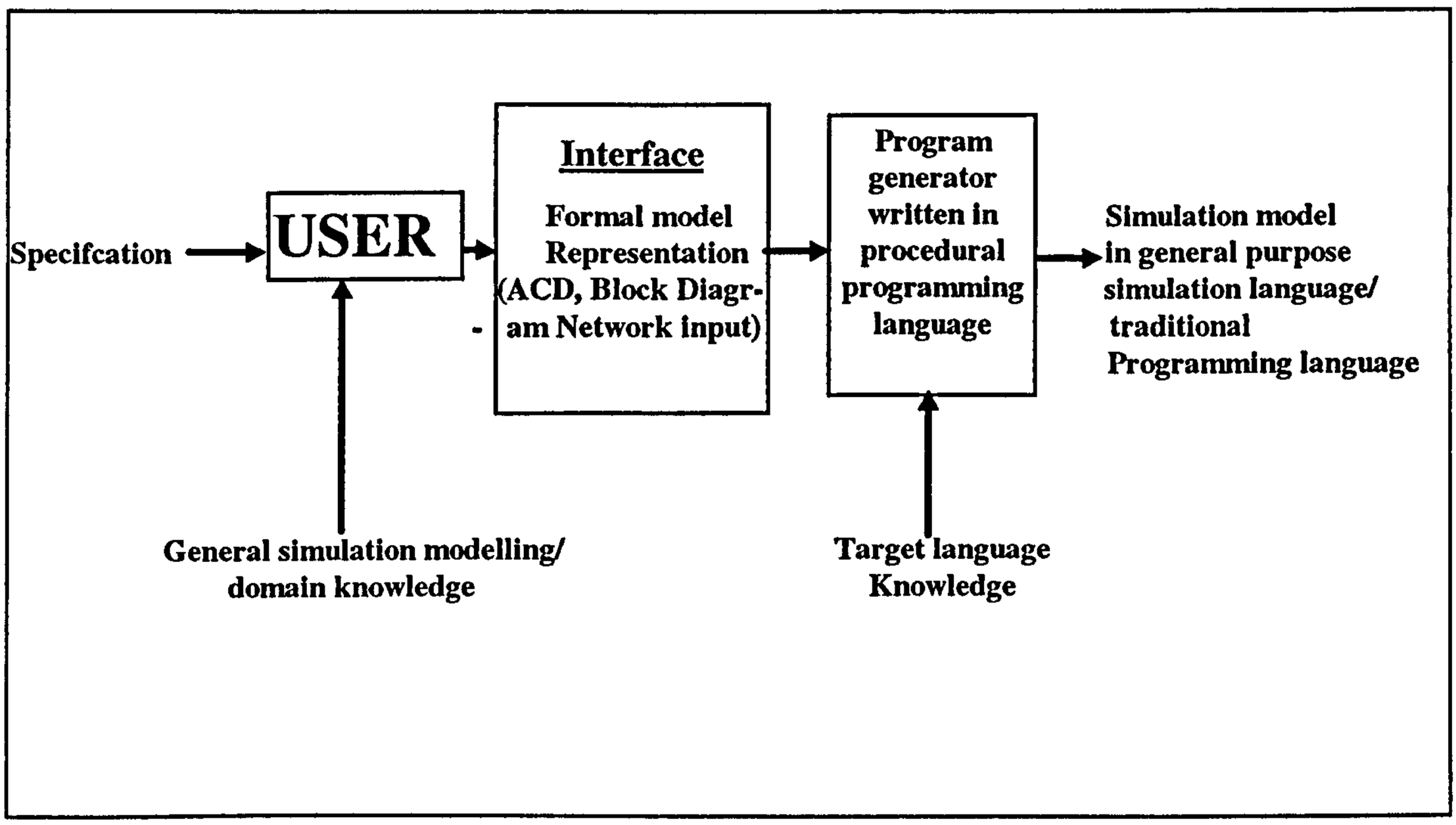


Fig 3.2 Components of Code Generators

Simulation code generators require as input a formal specification like an Activity Cycle Diagram (ACD) (Hills, 1965), or network input (Sinclair, Doshi, and Madala, 1985) and perform a translation to create the simulation model. An ACD (Carrie, 1992) of a simple machine shop model is given in Fig 3.3. This is a very simple model and the meaning is self evident, with queues represented by ellipses and activities by rectangles. The construction of such diagrams can become complicated and labour intensive for the representation

of operators, load/unload stations, pallets and palletising operations, vehicles and their assignment, etc.

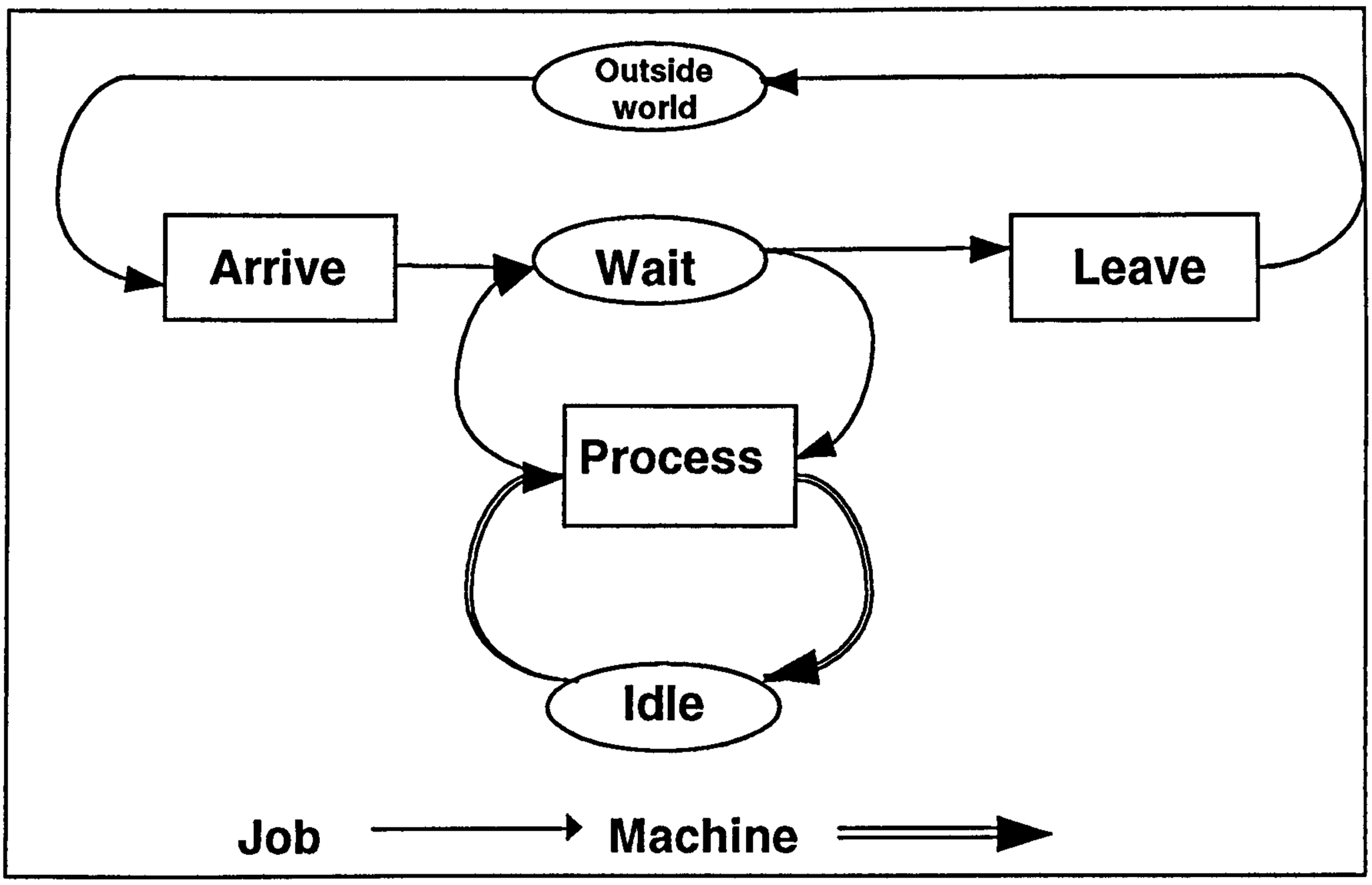


Fig 3.3 ACD of a simple machine shop

The formal specification is often entered interactively using:

- Questionnaires. These were used in some of the earliest systems and comprise a single form which provides fields for specifying all entities and their characteristics. In addition there are fields for entering all relationships between various entities. This method is descriptive and allows any model changes to be made easily, but has the disadvantage of requiring the user to understand what fields correspond to which characteristics.

- Dialogue Systems. In this method a number of sequential dialogues containing one or more questions are used to elicit the specification. In order

to reduce the number of sequential dialogues, the more advanced dialogue interfaces provide a number of data fields corresponding to the different characteristics of an entity, rather than having separate dialogues for each characteristic.

The main advantage of user interrogation is that it removes from the user any burden associated with writing the specification in any form of descriptor language. It is also the most natural method resembling the relationship between the domain experts and the model developers. The main disadvantage is that unlike with descriptive specification methods for long specifications there will be a large number of sequential dialogues needed for model entry making it difficult to keep track of what has been entered. Also dialogue interfaces are disadvantaged by their inability to allow the user to backtrack to earlier dialogues to change input.

- Graphics Interface. These are used to specify the layout of the model and are used in conjunction with one of the above specification methods, so that the characteristics of the components of the model can be entered.

The main method of translating the formal specification into a simulation model is ***procedural*** and involves writing a general purpose program that maps the specification into the lower level target language representation. This type of translation system is implemented using traditional general purpose computer languages like FORTRAN or PASCAL. Other translation methods employed in the general field of automatic programming are deductive and transformational; for additional information on this area consult Green (1969), Waldinger and Levitt (1974), Kowalski (1977), Clark and Sickel (1977), Darlington and Burstall (1973), Martin (1974), Lenat

(1975), Biggerstaff (1976), Fickas (1985), Balzer (1981), Manna and Waldinger (1980), Partsch and Steinbruggen (1983), and Cheatham (1984).

The majority of the generators produce models in high level programming languages or activity based general purpose simulation languages, although there are a few systems that generate models in the process oriented general purpose simulation language SIMAN. The only generator concerned with an event based language is Express (Shanehchi, 1985) for See-Why and is merely a front end for writing the FORTRAN code responsible for executing events; the events themselves have to be defined by the user, requiring substantial simulation expertise for the task.

3.3.2 Examples

The development of code generators is illustrated here with a few typical examples.

A system that uses a questionnaire as the specification method is MIDSEM (Davis, 1976), which has two stages for model generation. The model definition and the model generation. The system provides 3 model definition modules, each of which provides an interactive questionnaire representing a particular modelling scheme. The 3 questionnaires require information about the entities, activities and queues of an ACD, events as used in SIMSCRIPT, and information about bound and conditional events in the style of the ALGOL-60 Elliot Simulation Package (ESP). The system runs models in various forms; for example the Simulate module is an on-line simulator which runs the model directly from the input file, whilst the Model Generate module produces syntactically correct simulation language statements for the two phase or

three phase versions of ESP (this has a control structure similar to SIMSCRIPT).

DRAFT/GASP (Mathewson and Allen, 1978) is a member of the DRAFT family of code generators and employs user interrogation for eliciting the system specification. The function of DRAFT is to produce a compiler executable model. The background for this work is that of of Dr A.T. Clementson who designed a generator (Clementson, 1982) of this type for ECSL. The project has been extended to support a number of languages other than ECSL such as SIMON (II), SIMON 75, ACSL, SIMSCRIPT and FORTRAN, emphasising the point that the specification method is independent of the target language. The models are described using an Activity Cycle Diagram, requiring the user to have simulation expertise in order to develop the diagrams, but no target language knowledge is required. The advantages of this approach include clear problem description through the use of diagrams, and speed of code generation. The style and error free format of the generated programs contribute to these advantages. The translator for DRAFT/GASP is written in FORTRAN, in the form of a coded algorithm which transforms the logic of the Activity Cycle diagram into the code of the simulation language. The use of a questionnaire can be a long and tedious process when specifying a large system, and further work on the DRAFT system has resulted in the development of a graphics module called DRAW (Mathewson, 1985) and SSIM (Mathewson, 1985) which uses spreadsheet techniques as an alternative method of input.

Subrahmanian and Cannon (1981) developed a code generator written in PL/I which generates SIMSCRIPT simulation models. The model is described using a high-level descriptor language (very high-level language), in contrast to

previously mentioned systems which use questionnaires or dialogues, and allows the definition of model components, active and passive variables, component interactions, output facilities, terminal conditions, variable input and initial state of the model. This description is then parsed to produce a standard model description, which is independent of the target language, specifying all relationships between variables used in the model. This standard description is finally translated to produce the SIMSCRIPT code.

A system called Autosim (Paul & Chew, 1987) produces models in Pascal[7]. The system is written using Turbo Pascal and supported by LIBSIM, a library of Pascal subroutines. The model specification is via an ACD through interrogation, and each answer is used to guide subsequent questions. This specification is then converted into an interactive data file, referred to as the formal model specification. The specification requires the names of all temporary and permanent entities in the system; also life cycles for temporary entities have to be defined as sequences of alternating queue and activity names, together with their source-sink queues. After these cycles are specified the system analyses the specification and provides a review. This allows the user to check that he has specified the system he intended. The next part of the specification requires definition of the queuing disciplines for the queues defined during the life cycle description. The final part of the specification is obtained by interrogating the user for the duration of all activities, the arithmetic of any entity attributes, and the initial conditions.

---

[7]This system and the ones in the previous and proceeding paragraphs(Raczynshi,1990; Shearn1990) generate models in high level languages(Pascal) in contrast to the majority of generators which produce models in general purpose simulation languages.

The majority of generators require the editing of the generated program in order to model complex requirements not covered by the specification language. Such edits, external to the system are usually not easy and will have to be re-entered if there is any modification to the original model description.

This problem was addressed by PASSIM (Shearn, 1990) so that editing of the generated program would be virtually eliminated. When necessary, rather than editing the generated program, the system allows user written Pascal code that comprises the edits in the generated program to be included in the model description. There is also an option for the generation of a program from a current or previously defined specification.

The systems so far described use textual interfaces for specification purposes. However a number of other generators use graphical interfaces for specification. GIST (Sinclair, Doshi, and Madala, 1985) is one such system which allows the specification of Extended Queuing Network (EQN) models, through a graphical or textual interface. It produces executable images to simulate a model of the real world system. The GIST system is based on CSIM, a run-time support environment for discrete event simulation. It provides two interfaces, GUIDE (Graphical User Interface and Dialogue Editor) and TIDE (Textual Interface and Dialogue Editor). GUIDE allows the specification of EQN's by the use of graphics to define the model, in which icons from a set representing the various object types are connected together. It is particularly useful because EQN models are naturally represented by a network of objects and their interconnections. The object types in a GIST model are SOURCE, SINK, ALLOCATE, DEALLOCATE, DESTROY, FORK, JOIN, PROBE, SWITCH, QUEUE, SERVER AND QSERVER; each of these object types has a unique dialogue window allowing specification of object

names, classes of objects and distribution parameters. A complete description of these object types together with an example session is provided in the paper "A Graphical Interface for Specification Of EQN models (Sinclair & .Madala, 1986)".

A variation on the previous system is the Queuing Model Generator (QMG) (Raczynshi, 1990) which makes it possible to define a queuing model without any programming. It comprises a menu-driven layout editor, which permits the user to define the model by placing blocks and interconnection lines to depict all possible model events and the model structure. The editor performs a preliminary verification and does not permit obviously inconsistent diagrams to be created. After the diagram has been completed the system asks for the corresponding probability distributions and parameters. A Pascal program is then finally generated, compiled and executed.

3.4 Contribution of AI

It is stated (Shannon, Mayer and Adelsberger, 1986) that "Not only does simulation need AI technology but also ES (Expert Systems) badly need simulation. If one examines the current uses of AI and ES, it immediately becomes apparent that few of these uses have a time domain because the AI experts do not yet know how to handle time. One of the real strengths of simulation is its ability to forecast or project events and effects through time. The ability of Expert Systems will remain limited until the technology of simulation is incorporated and exploited."

The main contribution of AI to automatic simulation program generation has been in the areas of improved interfaces, knowledge representational schemes

and knowledge based translation methods. The attempt at improved user interfaces was in some way motivated by the desire to make communications with computers more natural. This involved research into natural language processors, and the use of such processors for simulation program specification; this was deemed attractive because it is often the most natural method of representing on paper a specification and provides vocabulary, informality and syntax.

There were other advantages of natural language processors like ease of use, with little training to become familiar with input requirements, and reduction in time consuming debugging and syntax checking steps because the post-processor ensures that the syntax of the simulation language is fulfilled. Also natural language descriptions are usually most compatible with the user's logical representation of the system.

However, a number of disadvantages of the natural language systems emerged:

- only successful when processing constrained input, meaning the user is limited in the structural variation of sentences he can use.
- reduced flexibility with skilled users finding them cumbersome.
- for modelling complicated systems, the size of the specification will often exceed the size of the generated program.

These disadvantages did not, however, prevent its application in further research as part of hybrid specification methods comprising natural language,

and dialogue and/or graphical interfaces. This had a knock on effect of accelerating research into alternative methods like dialogue and graphical interfaces.

The other main contribution of AI to simulation has been in the use of advanced knowledge representation and inferencing techniques. This resulted in the encoding of domain, simulation and target language knowledge explicitly in AI representational schemes like facts, frames, semantic nets and rules. For most of the automatic modelling systems of the next section, the target language knowledge is predominantly SIMAN, whilst the domain knowledge varies from queuing systems to electronics component manufacturing. The domain knowledge includes the concepts of the domain and the interrelationships between these concepts, and is required to infer how to choose a solution from the specification.

The advent of advanced inferencing techniques provided a more flexible means of model translation than the procedural methods used by the code generators. It provided a means of applying rules (general simulation and target language knowledge) to facts (the data structure representation of the model specification) to produce new facts (the simulation program).

3.5 Automatic Modelling Systems

3.5.1 General Approach

As Artificial Intelligence (AI) evolved and became prevalent within the field of program automation so it started to influence automatic simulation programming resulting in the development of automatic modelling systems.

These used all the innovations from AI research such as improved interfaces, knowledge representation and inference. These systems in contrast to most program generators which contain only target (simulation) language knowledge, contain general simulation modelling, domain and target language knowledge. The main purpose of such systems is to improve the environment in which the simulation program is specified by the reduction in the amount of detail the modeller is confronted with, via a specification method more natural to the user's problem domain and way of thinking. The general structure of such systems is given in Fig 3.4.

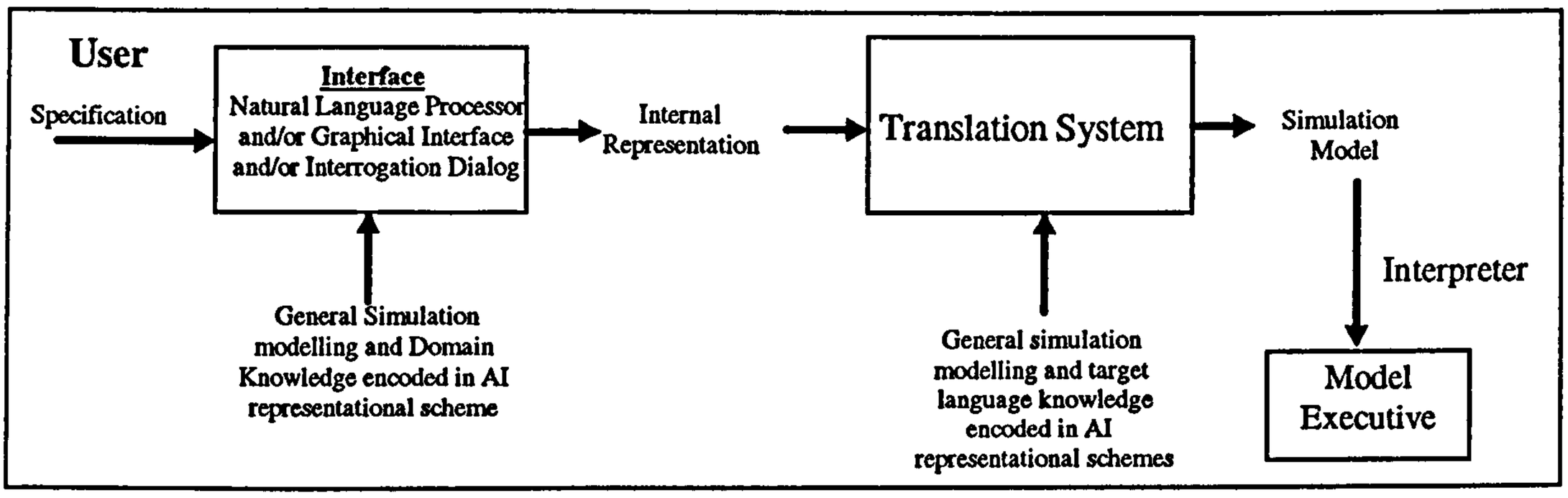


Fig 3.4 Structure Of Automatic Modelling Systems

The specification methods for automatic modelling systems include natural language, dialogue system, and a combination of a graphics interface and user interrogation. They all use AI techniques for knowledge representation and translation and the majority of the models are generated in the process oriented general purpose simulation language SIMAN.

3.5.2 Examples

The development of automatic modelling systems is illustrated here with a few typical examples.

A automatic modelling system by Ginsberg, etal (1965) can be used to create simulation models of job shops through the selection of options from a questionnaire. This questionnaire defines the scope and structure of all job shop simulation programs that can be constructed by the system. The questionnaire comprises an explanation booklet and an answer sheet requiring information about resource configurations, job characteristics, decision rules and probability distributions. The program generator, when these choices have been made, translates them into a SIMSCRIPT simulation program. The translator utilises a statement list, written in Pascal, which defines all statements used in SIMSCRIPT the target language. It also uses a set of decision tables to provide a link between the questionnaire choices and the statements in the statement list. The simulation program is produced via the application of the statement list using the decision tables.

A system developed by Murray and Sheppard (1986), referred to as an Intelligent Front End (IFE), generates code for simple queuing systems in the SIMAN general purpose simulation language. An interactive dialogue system is used to extract information from the user so that an internal specification of the system to be modelled can be formed and stored; it uses OPS83 data structures for representation. This internal specification is a description of the temporary and permanent entities together with their interactions. It also contains all information concerning model termination, transportation characteristics and statistical requirements. The OPS83 expert system building tool is used for implementation, with backward chaining and forward chaining rules used to create the internal model description and simulation model respectively. Model construction rules encoded in an AI language are used to transform the internal model specification into an executable simulation model.

Natural language processors have been used in a number of systems for simulation model specification. The earliest system using this approach was developed by Heidorn (1974) at the Naval Postgraduate School in the sixties, and automatically generates simulation code in the GPSS discrete event simulation language. The possible inputs to the system are restricted to a set of allowable syntactic and semantic constructs, which have to be learned by the user. This data structure is used to create a GPSS simulation program together with an English description of the problem, which is needed to verify that the user has entered the specification he intended. The sentences contain actions whose order is made explicit using conjunctions like "after", "when", and "before" or by the use of the adverb "then". If the order of the actions depends on status information, an "if" clause may be used for specifying conditions and an "otherwise" sentence for specifying an alternate action. Sentences are also required to specify the time between arrivals, the time required to perform each activity, the length of the simulation run, the time unit and the quantity of each permanent entity. The specification is analysed by decoding rules written in a rule based language and interpreted by a FORTRAN program, like a bottom up, parallel processing syntax directed compiler. The rules are phrase structure rules whose right hand sides consitute record building actions. The entity-attribute-value data structure is used primarily to describe the flow of temporary entities through the system, the actions that take place at permanent entities and the interrelationships between actions. The data structure comprises a number of records which represent physical entities (like cars and docks) and abstract entities (like actions and functions). The actions include the entry of an entity into the system and one or more of other actions such as wait, service, load and unload. For the GPSS model generation, these rules represent the allowable syntactic constructs of GPSS. The same process of inference is used to generate an

English description of the internal model description, where the grammar rules represent the allowable syntactic constructs of a limited English language.

A later project using the same specification method resulted in the development of an Expert System (Ford and Schroer, 1987) written in Zetalisp which translates a natural language specification of an electronics manufacturing plant into a SIMAN simulation program. The main components of the system are the natural language interface, a Simulation Writer, a Simulation Analyser, and the SIMAN discrete event simulation language. The natural language interface translates the description of the real world system into an internal representation called conceptual dependency and via a transformer generates the Lisp input commands for the Simulation Writer module. The Simulation Writer module takes Lisp input commands from the natural language processor and converts them into SIMAN code in the correct format and punctuation. The code once generated is reviewed and critiqued, and if any errors are found solutions are extracted from a patch library. The program is then run in the SIMAN simulation package, and if there are errors they are returned to the system and examined by the debugger. Corrections are then made to the code by the system and the reviewing, critiquing and debugging cycle is repeated until no more errors are found. The Simulation Analyser comprises expert rules for improving system efficiency, and recommends changes the Simulation Writer module should make to the code. Before these changes are made the user is given the option of modifying or accepting the proposed corrections.

Simtalk (Rummel, 1988) is a natural language interface for discrete event simulation which operates for a limited number of FMS. It operates as an

interface to a larger expert system which interacts with the user, models the manufacturing system, and generates simulation pseudo code. The system limitations includes 10 workstations with 10 identical servers at each station, 2 inspection stations with 2 possible outputs after inspection, user definable transportation between workstations and up to 3 products simultaneously routed through the system.

Haddock (1987) developed a system for FMS written in BASIC which takes a description of the system to be modelled and creates the simulation model in SIMAN. The three options provided allow the user to create, edit or run an existing model. After the model and experimental frames are generated they are interpreted and linked to comprise the complete simulation model. The system also provides options for output analysis by incorporating pre-programmed FORTRAN subroutines within the software structure of SIMAN, so as to provide assistance to the user not only in model development but also the analysis of the results. This additional option has resulted in the system being classified as a hybrid system since it comprises front and back end interfaces around an existing simulation system (in this case SIMAN). Work on this system was also referred to on the use of optimisation techniques (Bengu and Haddock, 1986) to analyse alternative system designs, investigating the use of search techniques to optimise mean number of entities in the system, mean time spent by them and system utilisation.

A system called the SIMAN Module Processor (SMP) (Endesfelder and Tempelmeier, 1987) generates operational simulation models of FMS by combining user-specific pre-defined modules using interactive pre-processor operations. The system is based on the realisation that separate model components can be combined or incorporated in existing models to form new models. The system operates by combining SIMAN modules to form the

model, and utilises the facility in SIMAN's to model sub-models. The sequence of problem specific adaptation includes the modification of probability distribution parameters and the consistent numbering of servers and queues Also, the system ensures that changes made in the model frame result in corresponding changes in the experimental frame. The process of developing a new model from pre-defined models is achieved by offering the user modules to be incorporated into the model, reading and interpreting the modules line by line and interrogating the user for relevant data as it progresses, and eliciting parameters of probability distributions. It is also possible for the user to develop his own modules and store them in the library, documenting each required input so that a novice user can, at a later date, follow the input commands to produce operable models. The main advantages of the system are that the configuration of new system variants eliminate the need to develop new models from scratch, the elimination of syntactical and logical errors has to be carried out only, once and the enlargement of the model can be carried out at any time. This system is unique because it incorporates both the concepts of software engineering, mainly model reusability and modularity, and automatic programming in producing simulation models.

Schroer (1989) developed a simulation assistant, using a similar approach to the SIMAN Module Processor (SMP) (Endesfelder and Tempelmeier, 1987) mentioned in the previous paragraph, providing a structured approach to modelling manufacturing systems via a set of pre-defined GPSS simulation macros, a user interface, and an automatic code generator. The library of subroutines comprise GPSS subroutines for each function in a manufacturing system including assembly, fabrication, inspection, and inventory. The user interface, written in turbo Pascal assists the user, through a dialogue, to define the system and its attributes. The output from the interface is a

specification file which is the input to the code generator; the latter selects the required GPSS macros from the macro library and automatically writes the GPSS simulation program. The experimental conditions are then elicited and the program executed, with any changes made via a number of options for modifying the problem specification.

SIMTOOL (Brazier and Shannon, 1987) is an interactive simulation tool written in Prolog for developing SIMAN simulation models of AGV systems. This is an automatic programming system equivalent to the Gong and McGinnis system (Gong and McGinnis, 1990) described earlier. The model, however, is specified using a graphical description of the network layout, comprising a directed graph of nodes and arcs which represent the AGV system. The nodes represent places where loading/unloading, vehicle queuing, route assignment, vehicle selection and vehicle blocking can occur. The arcs represent the routes an AGV takes while in the system by using a parent-child precedence relationship between connected nodes. The network, after it has been specified, is converted into data tables, which contain all node relationships and distances between nodes. Next, parameters concerning stations, AGVs and parts are specified using pull down menus, pop up menus, box menus and a series of questions. As an example, the part parameters include the number of part types, inter-arrival times, routing sequences, processing times for all parts on all machines, and the arrival and departure station names. This parameter information is stored in a file allowing modifications at a future date.

Gong and McGinnis (1990) developed a similar system which converts information as provided by the system designer or simulation user into a SIMAN (process interaction) simulation program for evaluating

manufacturing systems with AGVs moving along a uni-directional guide path network. The system is written in Quick Basic version 4.5, and it requires, as input, characteristics of permanent and temporary entities, together with their interactions. The translation system uses an internal representation comprising two types of modules; the 1st defines the system logic, whilst the 2nd determines the system control policies for determining decisions related to vehicle assignment, path selection, and machine selection rules. The model constructor, based on internal matrices, then recognises the problem specification and retrieves the system logic modules and the corresponding control policy modules from the program base to form the simulation model.

A System developed by Koshevis and Chen (1980) written in Golden Common Lisp provides an object oriented approach for the graphical representation of the real world system. From this representation SLAM code is produced and executed. It requires no prior knowledge of model building or programming, only knowledge of the system to be modelled and the functions of the icons in the system. The specification of the system is provided by placing these icons on the screen in their required positions, together with transfer lines to show the connections between nodes. The system then checks this representation for completeness by determining what components have to be added. It then prompts the user using interrogation for information regarding these components together with any missing information; for example, if a service node is present the system asks for the number of parallel servers, the duration of the service and queue information.

3.6 AI Based Simulation Tools

In this section we review how Artificial Intelligence aids simulation modelling, other than for automatic model generation. Although some of the systems are not entirely relevant to our research, they do provide an insight into specification and model construction methods

The first few systems we will describe are Knowledge Based Simulation Environments which attempt to reduce the cognitive distance between a design expert's description of systems and the descriptions supported by a simulation language. These are systems which store modelling knowledge using rules, frames, semantic nets and object oriented programming. The specification is usually acquired using a natural language, interrogation system, a graphics interface, or a combination of the three. These don't produce code but execute the model internally. The user can however monitor the execution or perform experiments interactively. Their purpose is to develop systems that cover the entire simulation life-cycle from model building, validation, refinement through to experimentation and statistical analysis. In addition they attempt to provide a far greater understanding of the phenomena being modelled by making models more comprehensible; they do this by representing the behaviour of objects in a manner more conducive to reasoning which is not possible in a procedural paradigm alone.

The systems comprise a set of objects which are standard within an application domain. For example in a manufacturing domain these objects will constitute machines, transport devices and storage facilities. A specific model is then created by specialising these objects and linking them via relations. The models are often

elicited by the use of domain specific knowledge acquisition systems (Fox, Seth, Baskaran and Bouer, 1986).

Research on such systems was initiated by the development of the SIMULA language which as an extension of ALGOL 60, and was deemed the first objected oriented language since it was geared to the representation of real world entities. The use of the object oriented approach, including message passing and encapsulation with inheritance hierarchy, has many advantages because it greatly simplifies the simulation of complex models resulting in highly comprehensible, modifiable and reusable models. Forward chaining rules can be used to represent the behaviour of human decision makers and are local to objects, whilst backward chaining rules in the form of messages can be used to achieve specific goals. In this way it is possible to mimic competently the behaviour of real world entities.

The Rule Oriented Simulation System (ROSS) is a large scale simulator (Klahr, Faught, and Martins, 1980) which incorporates AI techniques for modelling decision making behaviour in the domain of military air battles. It incorporates detailed and complex knowledge about the behaviour of real world systems modelled as objects. These objects are represented in a hierarchical manner allowing inheritance of properties as well as behavioural rules. The objects communicate by passing messages between themselves describing the actions that they take. In order to improve comprehension of knowledge and hidden embedded assumptions, ROSS uses rules to explicitly represent behavioural knowledge. To solve the problems of scattered and fragmented knowledge and to allow intelligible and efficient knowledge representation, object oriented programming has proved advantageous; for example an object called radar could have instances ground radar and airborne radar, which would inherit information from radar allowing the efficient storage and retrieval of knowledge. Since the

knowledge representation is made intelligible and efficient through object orientation it makes the retrieval of knowledge simple, and any modifications are inherited by all descendants of a class. The ability to alter behaviour allows the incremental addition and refinement of knowledge, resulting in ease of knowledge acquisition and system development thus, helping to keep the knowledge current; the modifications result in different simulation models, and ROSS provides a browser function which allows the examination of the simulator's knowledge. An explanation facility is provided to explain why certain events occurred and others didn't; this helps the understanding of how the simulator works, and the effects of certain behavioural rules.

A Knowledge Based Simulation (KBS) System developed by Reddy and Fox (1982) for organisational modelling is a Lisp based discrete event simulation package. The system is object oriented, developed as part of the Intelligent Management Systems (IMS) project. The model is composed of objects and their inter-relations, which match the user's conceptual model of the organisation. Models are stored as objects using SRL schemata, which incorporate an inheritance facility and allow the formation of networks. These schema contain slots which define limitations (facets) and status information. The slots also include rules to describe the behaviour of objects, and goals to describe performance criteria. The system informs the user whether these goals are met or not. Running the model involves the dragging of the schema network through time. The system is interactive allowing examination of the model and its behaviour. There are options for model creation, model alteration, graphics display, and monitoring and of run control. The modelling system provides a library of objects and relations which the user may use, alter and or/extend for his application. The schema for these various objects and processes are arranged in a hierarchy, where each schema may inherit slots and values from schema higher

up in the hierarchy. These modelling libraries are created from SRL schema representing various objects, processes, behavioural rules and scheduling algorithms for specific problem domains. Once these libraries are created a model is formed by combining relevant schema from the library.

SIMKIT (Zalevsky, 1988) is a frame based system, which represents real world entities as objects. The models are developed by SIMKIT, which sits on top of the KEE Expert System Shell, and are executed on a LISP machine. Each machine or product is represented by a separate frame containing information on its relationship to other frames and its unique and inherited characteristics. The simulation model comprises instances of objects from several libraries, which are hierarchically linked, becoming more specific as they move towards the model level. In these libraries there is a subclass structure, with the lowest level of the hierarchy containing the specific machines.

The Simulation Environment System SES (Adelsberger, Pooch, Shannon and Williams, 1986) was developed in order to allow more natural model representation and experimentation. The system employs object oriented programming to elevate the model and experimental file to a higher level of abstraction than is possible with traditional simulation languages. It combines these object oriented concepts with knowledge based systems and simulation to create an user friendly interactive environment. The system also includes a conflict resolution mechanism, and a goal driven run-time environment for advanced problem solving. The simulation model and the experimental files are treated as objects, which are created or modified using a graphical interface, template/menu input and natural language dialogue, or by a specification language. These objects correspond to modular components of the real world. The behaviour of these simulation model objects describe the behaviour of the

modular components of the real world in response to various inputs. The interaction of these objects is by the passing of messages, which describe both functional and relational actions. The object oriented approach effectively represents a correspondence between simulated and real world objects. Also once an object is created, it serves as an abstraction for future refined objects. This results in an hierarchy of objects with inherited properties and behaviours representing the real world to be modelled. After the model and the experimental frame are created, they are checked using conflict resolution for consistency and completeness. The run time environment provides verification of simulation results via run time displays.

An Expert System by Shannon (Shannon, Mayer and Adelsberger, 1985) for the manufacturing domain stores a number of models in a database. These may be complete models or sub-models representing the operation of various real world objects. The user then specifies the system and the experiment, allowing the system to search the database for the appropriate model. If a suitable model is not available, the system constructs a model by incorporating sub-modules into a main model. In addition, a model generation facility is available; the user specifies the manufacturing system allowing the system to automatically generate the model code. This would require the specification of the plant layout via the placement of icons corresponding to the different manufacturing facilities; additional information like station numbers, speeds of transportation devices, and buffer sizes would be elicited by interrogation. In this system a corporate data base could be set-up to aid user input, containing information on all parts to be manufactured i.e. machine setup times, machining times, load/unload times, etc. The user can then specify the part (number, group), batch sizes, arrival rates and dates, proposed schedule and historical record, allowing the system to search the

database, extract the relevant information and associate it as attributes of the entity.

A discrete event simulation package T-Prolog (Futo and Szeredi, 1982) extends the traditional approach to simulation by the use of automatic problem solving. This involves backtracking in time, and the automatic modification of models depending on logical deductions. The models are defined using first order predicate logic using horn clauses. There is a provision for interrupting the simulation run so that the model can be modified by the addition or deletion of logical statements from the original set of clauses. The system can be forced to return to a previous state for trying possible alternatives, if there is deadlock and a goal cannot be satisfied. The main problem with the package is that the user must be proficient in predicate logic in order to define the problem, and it is therefore not an automatic programming approach. The system combines the time handling primitives of simulation and the symbolic processing capabilities of Artificial Intelligence into a Prolog superset. It allows specification of multiple model parameters and goals; the interpreter then executes the model and tries to find the first parameter set that matches the goals. The simulation approach is called AI based since the behaviour of the model is examined at various points during the running of the simulation by the use of control conditions. Also the structure of the model can be altered automatically according to rules invoked depending on the state of the system. The role of time is essential since the conditions for synchronising the processes of a real system are time dependent.

A system called PROSS (O'Keefe and Roach, 1987) is a Prolog based implementation of GPSS and uses a hybrid approach, where rule based scheduling polices reside alongside a process oriented simulation model. This is achieved by linking Prolog rules to a process description simulation model. The

reason for developing this hybrid approach is that Prolog's AI based knowledge representational scheme expressed as rules is a convenient way of representing many scheduling algorithms. Further rules allow the representation of heuristic knowledge; thus a simulation model with a rule based component can contain complex heuristic rules.

3.7 Discussion

From this chapter we can conclude that automatic simulation programming must be domain specific containing knowledge of the specific problem being addressed. Also it must consider aspects like efficiency, general programming knowledge and constraints imposed by the hardware and the target language. They cannot be expert in every domain because this is beyond the current scope of AI. The simulation code generators have a wider scope than automatic modelling systems, since they are merely translators for writing simulation language programs, with the domain knowledge provided as part of the model specification. However, they require a more formal specification (e.g. ACD) than automatic modelling systems which are tied to tighter domains e.g. FMS, AGVS, jobshop, etc.

Two possible methods (Barstow, 1985) of making automatic programming systems domain independent, together with an explanation of why these methods are unrealisable, are:

1. The first approach would involve having the program specification comprise a description of what the software is to do, together with a set of definitions that allow the program description to be understood (the domain knowledge). The problems with this are:

- The coupling of the domain knowledge to a single program will restrict its reusability. This is disadvantageous because the coupled knowledge may be applicable to a number of programs.
- Domain knowledge is more diverse than just object definitions, ranging from problem solving heuristics to expectations about the run-time characteristics of the data.
- It would be difficult for a computationally naive user to express the domain knowledge unless the system knew a significant amount already. Therefore it is advisable to separate the program specification and the domain specific knowledge.

2. The second method is where the domain knowledge is provided as part of an interactive specification process. Here the system is initially ignorant of the domain, with the user providing the domain knowledge during the process of specifying a program; after a number of programs have been specified the domain knowledge would have increased, thus solving the reusability problem mentioned in the previous case. The drawbacks of the method are:

- Again there is a requirement for a sophisticated user, although this problem can be overcome to some extent by having sophisticated users initially, with the naive users only using the system once there is considerable domain knowledge incorporated in the system.
- The main drawbacks that make this type of system unattainable are the problems with the organisation and structuring of domain knowledge, and that little is known about how programming knowledge and domain knowledge interact.

The specification method that is used is of great importance, and should support ease of use, conciseness, flexibility and incremental development.

There have been problems in some specification methods resulting in specifications being as long as the generated programs and almost as difficult to understand. This has sometimes resulted in a mix of a number of different methods being used. The knowledge representation schemes should also be chosen wisely and should be capable of representing domain, general simulation and target language knowledge in a concise and easily modifiable manner.

The initial aim of automatic simulation programming was ease of use and an improvement in productivity, and it has become allied to Artificial Intelligence research. Subsequently its limitations have been tied to those of AI, specifically that its application for substantial program automation has only been successful for narrow domains. This is due to AI being particularly successful in applying a depth of knowledge in a narrow domain as opposed to breadth. These limitations also apply to the earlier simulation code generators, which are effectively expert systems themselves since they encode simulation coding expertise within a procedural programming framework. Therefore, all simulation program automation systems thus far are restricted to certain domains i.e. simulation code generators to the simulation domain whilst the automatic modelling systems to specific real world system domains e.g. FMS, queuing system, AGV system, etc. It is also evident that they are even restricted in their capabilities within these domains. For example none of the systems have attempted to provide faciltities for specifying certain modelling intricacies (e.g. scheduling rules and labour priorities)

Due to these limitations in flexibility and ease of use, AI based modelling has shifted its focus from a strictly program generation task to simulation

environments that support the entire simulation development life cycle:- model building, validation, experimental design and output analysis

Chapter 4 Analysis of Generic Manufacturing Simulators

4.1 Introduction

Generic manufacturing simulators, as briefly mentioned in the previous chapter, were an improvement on special purpose simulators operating in limited domains, and attempted to provide greater modelling flexibility with little compromise in ease of use. This was achieved in some simulators by providing a programming language within a data-driven framework; alternatively other simulators were very large models with a large choice of input data options to instantiate it for a wide range of application. Examples of generic manufacturing simulators are WITNESS, PROMODEL, FACTOR/AIM, XCELL+ (Conway and Maxwell, 1986), MODEL MASTER (Bolin, 1986), and SIMFACTORY.

These simulation systems are geared towards the entire manufacturing domain, and not just specific types of systems such as FMS. They provide common elements for modelling the components, which are present in manufacturing systems, together with their characteristics. This is achieved via the provision of modelling modules which are used for representing, amongst other things, machines, conveyors, tracks, vehicles and parts. Certain modules, dependent on the particular language, provide some form of routing facility for defining the interactions between permanent and temporary entities. In addition, in some systems, there is the provision of options for the

manipulation of system variables and attributes; this can lead to greater flexibility but is achieved at the expense of ease of use.

They raised the model specification to a higher level of abstraction from traditional simulation languages by incorporating application domain (manufacturing related) knowledge, as well as substantial simulation knowledge in the specification language used as the user interface. Their development can be seen as the result of the influence of knowledge based and object oriented techniques, but they usually don't use the tools and formal techniques of either. Real world objects and their behavioural components are defined as aggregate building blocks within the systems, and the user provides data to instantiate them for a specific model. Hence, they are often referred to as data driven modelling tools; however, to obtain the full flexibility that may be needed to model any complex system, the simulators generally do offer the option to add user defined behavioural knowledge in a free format programming style, either within the specification environment (e.g. as in WITNESS) and/or by linking to a routine written in a third generation general purpose language (e.g. as in PROMODEL).

4.2 Simulation of manufacturing systems

A manufacturing system is primarily concerned with transforming the shape and form of parts (material) with the aid of various processing units (machines). The presence of a material or machine focused view of the world in

simulation languages was first mentioned by Tocher (1965), who concluded that all simulation languages to date had one or the other as the dominant entity type in the way models are constructed. Some of the early simulation languages and their dominant entity type is shown in Fig 4.1. In a material based view of the world, machines (and other resources) play a subordinate role, simply serving material/parts, while the logic governing the behaviour of the manufacturing system is contained in the definition of the latter in the form of a process plan (or route). In a machine based view of the world, such logic is contained in the definition of the machines or, more precisely, machining activities. In practice, however, more neutral terms for modelling constructs were used in the systems since they had applications in fields other than manufacturing. The material based view is synonymous with the process interaction approach, with the process routes of temporary entities as dominant features in the model. The activity based systems, with their modelling approach based on the processing activities which are responsible for changing the status of the entities, coincide with the machine based view of the world. SIMULA, as an object oriented simulation language, can combine both material and machine based views of manufacturing systems; interestingly, as discussed in section 4.4, the presence of such a hybrid approach can be detected in at least some of the manufacturing simulators that were developed much later.

Simulation Language	Dominant type of Entity
GPSS	Material
SIMPAC	Material
SIMSCRIPT	Material
SIMULA	Materials and Machines
CSL	Machine
ESP	Machine
GSP	Machine
MONTECODE	Machine
SIMON	Machine

Fig 4.1 Early simulation languages and their entity focus

A manufacturing system, of course, uses a number of other resources, primarily labour, and various types of transportation units to aid material flow. These resources, together with material and machines, are the physical elements of the manufacturing system, i.e. the structural components needed to define it. For its full definition, the logic governing material flow and interaction between various resources will also need to be represented, which constitute the behavioural components of the manufacturing system; these could simply be the priority rules that are used to determine the next part to be processed by a machine, or complex control systems governing the behaviour of Automated Guided Vehicles and flexible numerically controlled machines.

4.2.1 Structural Components

4.2.1.1 Parts and Machines

As already discussed, parts and machines (processing stations) are the two key elements in any manufacturing system. How they and their interactions are treated define the central characteristic (world view) of a simulation system. In a material based process interaction view of the world, the active elements are the parts and their interactions with machines are defined primarily through process plans; a machine based activity view of the world treats the machines as the active elements, with their interaction with parts defined by the changes in the part status brought about by the machining activities.

4.2.1.2 Materials Handling

Materials handling systems are employed to move materials, WIP, completed parts, tools, scrap etc. from one point in a manufacturing system to another. Materials handling systems can be manual like cranes, trucks, forklifts, etc. Alternatively they can be mehanised and automated like conveyors and AGVs. The discussion here concentrates on automated materials handling, because of their complex control requirements and requirement for greater flexibility in the modelling tool.

There are usually two classifications of conveyors: fixed conveyors, which index parts a position at a time, include belt, chain, tray and trolley;

accumulation conveyors, which allow queuing, like roller and tow line. A conveyor system can be a number of conveyors connected together, or a single conveyor comprising a number of segments. A conveyor system could be viewed from a machine based focus by visualising each segment as a machine. However, there are inherent problems with the approach, specifically that the travel time of a part is indeterminate because it is affected by the delays suffered by parts that precede it on the conveyor. The material based view is also inadequate since it views machines as being idle or busy, whereas accumulating conveyors are always busy.

An Automatic Guided Vehicle (AGV) system is a more flexible method of moving materials and components from one point to another in the factory. It is possible for track systems like AGVs to be represented with the machine based view by having different sections of the track modelled as different machines with their own control rules determining the movement of the vehicles; there would be problems with modelling AGVs with a material based view since they are resources with complex control systems/rules governing their individual behaviour and not just passive elements supporting the operation of parts.

However, making materials handling systems conform to machine based view is a totally unsatisfactory approach from the viewpoints of terminology and conceptual integrity, and most simulation software provide specialised modelling elements for representing them. These elements provide means for

representing vehicle routes, track layouts and dimensions, conveyor dimensions and operating characteristics, etc. However, it should be noted that the inclusion of these additional modelling elements do not change the basic world view of the simulation software, since in material based systems the behavior of the materials handling system, such as its allocation, movement and release, will be controlled by the part (material).

4.2.1.3 Buffers.

A manufacturing system would normally contain some work-in-progress (WIP) at designated buffer storage locations usually, but not always, next to machines or processing stations. Such WIP is simply parts waiting (queueing) for one or more resources, and the buffers can be regarded as passive elements through which a part passes during its life cycle. In simulation software with a materials view, buffer locations that a part visits would be contained in a process plan (note:-this differs from the traditional view that a manufacturing engineer normally takes of a process route, which usually only contains the processing stations the part visits). In a machine based view of the world, the storage locations are simply treated as the queues from which parts are drawn by the machine or to which a part is sent by it on completion of the required manufacturing operation.

4.2.1.4 Labour and other resources

Labour is often required by a machine for its operation, setup and repair. It is also required in less automated systems for transportation. The majority of manufacturing simulation software provides special modules for modelling labour and allocating them to processing operations within a simulation, but they and other similar resources are treated as passive elements serving a part (material based view) or machine (machine based view).

Inspection is traditionally performed manually and is highly time consuming and expensive, which has lead to the development of modern automated inspection systems utilising sensors, optical methods, probes and specialised machine tools. Inspection operations are usually modelled in exactly the same manner as machining activities, which may or may not require the use of labour (inspector).

4.2.2 Behavioural components

In addition to the physical constituents or structural elements that define a manufacturing system, the behavioural elements, which relate to the interaction of different structural elements, also need to be considered. This most commonly relates to the interaction between materials and machines, and in the simplest type of production control refer to the use of a number of standard queuing or priority rules to control material flow through a plant.

These rules, however, take into account only the local information related to the operation for which the material is queueing. It is desirable in some cases to base decision making on the global system state i.e. scheduling an operation based on the status of subsequent operation stages.

The ease with which scheduling rules can be included in a simulation model is also dependent on the tool used for its implementation. For example in some systems they can be easily modelled (at the cost of flexibility) because they are built into the simulator and merely have to be selected. In other systems they have to be programmed resulting in greater behavioural modelling flexibilty at the cost of reduced ease of use. However, some practitioners (M. Da Silva and Bastos, 1986;Bhattacharyya, Roy and Huang, 1989) have commented on the lack of flexibility of traditional simulation software for modelling more complex decision rules based on overall system status; in particular, it is generally difficult to represent any but the simplest priority rules for allocation of labour since, in both material and machine based world views, labour is modelled simply as a passive element serving parts (material based view) or machines (machine based view). To overcome the problem with inflexibility the authors have proposed a knowledge based hybrid approach, which combines a generic simulation model with a declarative knowledge base containing the decision rules.

4.3 Classification of manufacturing systems

Historically the development of simulation software was governed by their suitability for modelling certain types of manufacturing systems. There are a number of different types of manufacturing systems which are distinguished by their production volume, product variability, layout and level of automation.

4.3.1 Production Lines.

The layout of production lines is product based, where the processing and assembly resources are placed along the line of flow of the product. These systems process materials which have very similar characteristics, with a process route that is usually fixed and restricted by a non flexible material handling system (rotary worktable, fixed transfer mechanism). Therefore, since the process route is relatively fixed, it is reasonable to conclude that the machines determine the sequence of operations to which the parts are directed; the process route of a part can be implicitly defined in the model with the starting conditions of machining operations and actions taken at their end. The complexity of interactions between material and machines would often lie in complex, individual control rules associated with each machine. Thus mass production systems of this type have a natural link with simulation software which have activity based world views.

4.3.2 Jobshops

In a jobshop type of manufacturing system, typically a large number of jobs/orders with varying characteristics and widely different process plans are produced, on general purpose machines. Due to the general simplicity of the machine characteristics and the diversity of the process routes of parts, jobshops have a natural affinity to the material based view of the world, and could be regarded as providing the basis for the development of the process interaction approach as one of the main paradigms in simulation software. In such systems the process route (or plan) provides the main element for modelling the machine- material interaction, and using it as the basis for modularity in the model makes addition or alteration of the route relatively simple; in a machine based view, this would be more difficult since the logic of all the affected machining activities will need to be altered.

4.3.3 Batch Production Systems.

These systems produce the same product in medium lot sizes and aim to satisfy continuous customer demand. They achieve this by building up inventory of a specific product before moving onto another. When the inventory of the first product becomes low it returns to produce more of it. The aim of providing a degree of product variability whilst maintaining a sufficient level of production volume has meant that the machine tools used are again general purpose with alterations made for higher production rates. The

systems are often arranged in a process based layout, where machines are arranged in groups according to the manufacturing process. In batch production systems, where batches have different characteristics, as with job shops, the process routes vary making the dominant entity the material.

4.3.4 Flexible Manufacturing Systems.

Concepts of flexible manufacturing system (FMS) were developed to bring many of the benefits of mass production to small volume batch production environments. Flexible CNC machines are used, with fast changeover times to produce a large variety of products or variants of the same product, and making them easily adaptable to changing market requirements. Automated materials handling systems (e.g. AGVs) were also introduced to speed up material flow to keep work-in-progress (WIP) low and, thus, make the overall manufacturing system more responsive to customer demand.

Since these systems are highly flexible, with the ability to produce a wide variety of parts, the process routes are highly variable, making the material the dominant entity. However, since these systems often incorporate complex materials handling systems-to aid flexible routing, the material is not the only dominant entity; complex control rules governing the behaviour of physical resources (permanent entities) are often characteristics of such systems, making it necessary to model their activities in some detail. The early process interaction simulation languages were not very good at modelling these newer

material handling entities. However as simulation software evolved, they were accounted for by the incorporation of special modelling facilities (statements in general purpose simulation languages and modules in generic manufacturing simulators). The machines themselves can have complex control rules governing their behaviour. Often a machine has little or no buffer attached to it and, instead of playing a passive role of working on the next part in the buffer (according to some priority rule), its control system may call on the next part to be routed to it; the machine based view is more appropriate in such cases. Hence, an ideal modelling tool may need to combine both material and machine based views, and incorporate special facilities for representing complex material handling systems.

4.4 The World View Of Manufacturing Simulators

Manufacturing simulators were developed to enhance model specification to a higher level of abstraction by incorporating domain knowledge in the simulation software. There is no particular evidence in the literature to suggest their development was based specifically on existing simulation modelling paradigms i.e. event, activity and process interaction approaches. Instead, the manufacturing simulators were developed with modelling constructs to represent real life objects (parts, machines, AGVs, conveyors, etc.) and their behaviour. An analysis of the systems, as illustrated in this section, also clearly reveals that each was primarily developed with a world view of particular type of production system (mass production, jobshop, etc.).

This, however, inevitably provided them with either a dominantly material or machine based view of the world. Three well known simulators (WITNESS, PROMODEL and FACTOR/AIM) were selected as representative systems throughout the study, and were examined for their inherent world views. The three were, in fact, selected for differences in the style and structure of model development that characterise each, and could be considered as typical of a wide range of manufacturing simulation systems that are now available.

4.4.1 WITNESS

WITNESS provides the user with a menu based form and Graphical User Interface for specifying a model (examples of the forms can be found in Appendix A), and the specification language creates a list file of the model details; the latter is the internal data structure that the simulation system within WITNESS uses. The focal point of the specification for core manufacturing activities in WITNESS is the definition of machines, which is used to specify both the resource and the machining activity (and, hence, the interaction between it and parts). This gives WITNESS similarities with an activity based structure (machine view). Very early versions of the system did not offer 'part route' as an aggregate modelling element, suggesting the origin of its world view of manufacturing was based on mass production lines with little variety in products; later versions, however, included 'part route' as a modelling element.

The route element within the part definition module, if used to define the sequence of operations a part passes through during its life cycle, gives WITNESS a material view of the world similar to that in the process interaction approach. A simple machining operation comprising: part arrival, delay in queue, machine seizure, delay by operation time and machine release, would be represented by the route entries in Fig 4.2. If however a route is not used, the sequence of operations can be defined by the use of output rules in the part, queue and machine definitions. For the single machining operation defined above, the output rules are:

PUSH TO QUEUE For Part

PUSH TO MACHINE For Queue

PUSH TO QUEUE For Machine

STAGE	DESTINATION	R_SETUP	R_CYCLE
Stage 1	Queue	0	0
Stage 2	Machine	0	10
Stage 3	Queue	0	0

Fig 4.2 Single Machining Operation in WITNESS

Since a WITNESS model has the characteristics of both the material (process interaction) and machine based views (activity scanning) it can be thought of as a hybrid (activity/process) simulator.

It is possible to dictate the order in which the WITNESS executive attempts to output parts from blocked elements and input parts to idle elements, whenever a change occurs in the model. This can be achieved in one of two ways:

- By event: Every time a change occurs in the simulation (that is, every time an event takes place) WITNESS tries to obtain work for elements which are idle. This is the two phase method usually employed in the process interaction approach.
- By time: Just before WITNESS completes actions associated with all activities that are due to finish at the current simulation time, it updates the clock and then attempts to unblock any elements. This is a three phase approach, nearer to that used in an activity based system.

Time mode allows greater account to be taken of priority given for allocation of resources; however, it does not suffer from the degradation of computational efficiency as in the normal three phase activity based approach since a full activity does not take place. Time mode also allows the more realistic modelling of queuing conveyors, as the next part to be pulled from the conveyor is ready after one conveyor cycle in time mode (whereas in event mode the next part is only ready after two conveyor cycles).

4.4.2 PROMODEL

The internal model representation in PROMODEL is in the form of a number of tables (see Appendix A for examples of tables) to which the user can add data values either directly or through a DOS Interface. The focal point of the specification of core manufacturing activities is the definition of parts, which is used also to define the process route of a part that represents its interaction with resources. This is similar to the world view contained in process interaction (PI) based simulation systems (e.g. SIMAN[Pegden, 1985]), which has its root in batch production systems and jobshops with a variety of parts. The programming part of the model is restricted entirely to the ***routing*** module which specifies the process plan for all materials. A single machining operation in the PI approach (Fig 2.8) comprises the sequence: part arrival, waits in queue, seizes machine, waits until completion time elapsed and releases machine. This representation is equivalently represented in PROMODEL as shown in Fig 4.3.

Part	Location	Operation (min)	Output part	Next location	Condi-tion	Qty	Move time (min)
p1	queue	7	p1	m1	0	1	conveyor
p1	m1	10	p1	queue	0	1	conveyor
p1	queue	5	p1	Exit	0	1	conveyor

Fig 4.3 Single Machining Operation in PROMODEL

There seems to be no tendency towards the machine based approach and, therefore, activity based world view, since the machines signified by location pay no part in the modelling process apart from processing points within the routing module. They don't even have to be defined in advance of their use within the routing module, which is the case in WITNESS. The only other roles played by the physical elements likes machines is in the definitions of their capacities and breakdowns. Explicit modelling constructs of interrupting activities, like breakdown of machines, that most manufacturing simulators provide, however, gives an element of an activity structure; in a true PI system, interruptions have to be modelled by the user as part of a process route (using various artificial methods, e.g. using a dummy 'part' to occupy a machine while it is broken down).

From an execution of the model, an analysing the trace of the simulation run, it can be confirmed that the program executes the process plan by scanning idle resources and by the use of a sequence of actions like:

- **PART** begins move from **LOCATION A** to **LOCATION B.**
- Operation time for **PART** at **LOCATION B** is **TIME.**
- Operation time completed for **PART** at **LOCATION B.**
- Begin output logic for **PART** at **LOCATION B.**
- PART queues for output at **LOCATION B.**

4.4.3 FACTOR/AIM

The internal model representation in FACTOR/AIM is in the form of a DB2 database which is populated by the user through menu base forms and Graphical User Interface. However, data from Excel compatible files can be loaded directly into the database. Although machines and all the physical elements of the manufacturing system have to be defined first, a predominantly material based view exists in the system since the machine-material interactions are defined via a ***process plan*** module that specifies a sequence of job steps comprising of transportation (***move_between)*** and processing (***setup/operation)*** actions. A single machining operation is determined by a process plan comprising a number of jobsteps, as shown in Fig 4.4, with the processing time of 10 given in the Setup/Operation editor (Appendix A Fig 21) in the operation time field. Modelling constructs provided for interrupting activities, however, again gives an element of an activity structure.

Jobstep	Type	Description	Next
Js1	Setup/Operation	Queue	Js2
Js2	Setup/Operation	Process at M1	Js3
Js3	Setup/Operation	Queue	js4

Fig 4.4 A Single Machining Operation in FACTOR/AIM

The FACTOR/AIM executive would execute the following jobsteps in three stages, each containing a number of sub-stages. These are:

1. Start of jobstep event at time **t0**. Process jobstep **js1**

   •Load **Part 1** of order.

   •Allocating 1 unit of pooled resource **Queue**.

   •Allocation complete.

   •Schedule end of service for jobstep at time **t1**

   •jobstep js2 selected as next.

2. Start of jobstep event at time **t1**. Process jobstep **js2**

   •Load **Part 1** of order.

   •Allocating 1 unit of resource **Machine**.

   •Allocation complete.

   •Freeing one unit of pooled resource **Queue**.

   •Schedule end of service for jobstep at time **t2**.

   •jobstep js3 selected as next.

3. Start of jobstep event at time **t2**. Process jobstep **js3**

   •Load **Part 1** of order.

   •Allocate 1 unit of resource **Queue**.

   •Allocation complete.

   •Free one unit of Resource **Machine**.

   •Schedule end of service for jobstep at time **t3**.

   •jobstep js4 selected as next.

4.4.4 Other generic manufacturing simulators.

It is clear from the above discussion that both machine (activity) and material (PI) based views can be used to develop manufacturing simulators, and a hybrid approach can also be taken to give the user the choice of defining the machine-material interactions primarily through that of parts (in the form of a process route/plan) or machines or a combination of both. A number of other simulators were also examined with the help of their literature (brochures and research papers), and Fig 4.5 presents the dominant world view of some of the particular simulators. It should, however, be recognised. that all manufacturing simulators are hybrid in their nature at least to some extent, e.g. the way interrupting activities and transportation devices (e.g. AGVs) are modelled in a predominantly PI system have similarities with an activity based approach, while even the systems with a predominantly activity structure use a simulation executive that behaves more like that of a PI system. The key common feature amongst them are the real world objects and their behaviour, and this emphasis has led to a more pragmatic approach to be taken than the paradigms of the early simulation systems.

Simulator	World View
FACTOR/AIM	PROCESS
PROMODEL	PROCESS
WITNESS	ACTIVITY/PROCESS
XCELL+	ACTIVITY
SIMFACTORY	ACTIVITY
ARENA	PROCESS
MODEL MASTER	ACTIVITY

Fig 4.5 World view of Generic Simulators

4.5 Common Modelling Elements in Generic Manufacturing Simulators

From the investigation of generic manufacturing simulators, through direct use, published literature and vendor's brochures, it is also evident that a number of common modelling elements are provided by all of them. These elements allow the modelling of the commonly found components or constituents of manufacturing systems, together with their characteristics, like products, processing stations or machines, buffers or receiving areas, manpower or labour, transporters and tracks and finally interrupting activities (breakdowns, setups).

The investigation of generic simulators resulted in the classification in Fig 4.6, which shows the common modelling elements that are available to the modeller in a number of popular generic manufacturing simulators. The similarities are obvious and stem from the fact that they are all based on a common domain; in some instances slightly different names are used to represent similar elements (e.g. transporters and vehicles) and sometimes a generic name is used instead of a specific element with the same features (e.g. resources and labour).

Element	*WITNESS*	*PROMODEL*	*FACTOR/AIM*	*SIMFACTORY*	*AUTOMOD II*	*XCELL+*
Parts	Part	Part	Order	Object	Load	Stock
Process Plan	Route or INPUT/ OUTPUT rules	Routing	Process plan and Jobsteps	Process plan comprising operation sequence	Process System comprising procedures for: if then else logic; actions to take down, bring up resources; actions to choose processes, resources, queues based on their state	Processes
Machines	Machine	Location	Resource	Station	Resource	Workcentre
Buffers	Buffer	Location	General Pool	Queue	Queues+Order Lists	Buffer
Conveyors	Conveyor	Conveyor Section	Conveyor Section	Conveyor	Conveyor Section	Conveyor
Manpower	Labour	General Resource	Multi-capacity operator	Resource	Resource	Auxiliary Resource
Transporters	Vehicle	Transporter	Vehicles	Transporter	AGV	Carrier
Tracks	Tracks	Transporter Paths	Transporter Segments	Domain	Guide Paths	Path

Fig 4.6 The common elements within Generic Manufacturing Simulators

Each of the common elements provide input facilities for defining their characteristics, for example: the parts element will have means for specifying name, maximum number of arrivals of a part type, time elapsed between successive arrivals, time of first arrival, batch size; whilst the transporters element will have means for specifying characteristics like pickup speed, delivery speed, load time, unload time, capacity, etc. Similarly means will be provided for specifying the characteristics of the machines, buffers, conveyors, manpower and tracks.

4.4.1 Behavioral elements

The only elements that appear not to be common amongst simulators are those related to rules provided for job priorities and resource assignment. These rules may be in-built (FACTOR/AIM) or have to be written using an in-built programming language (WITNESS, PROMODEL, TAYLOR II (Nordgren, 1994)

For example:

- In WITNESS scheduling rules are programmed using ***Actions*** (for modifying variables and attributes) and ***input/output*** rules. A number of ***elemental*** in-built rules are available to the user, such as: WAIT, PUSH, PULL, IF, MOST, PERCENT, SEQUENCE, SELECT, BUFFER, FLOW, CONNECT, RECIPE, DESTINATION. Composite rules can be formed from these elemental rules, or user defined rules can be developed, using the

various programming constructs that WITNESS makes available. An example of a composite rule would be:

```
IF NPARTS(m1)>1
    least_slack_rule
else
    most_work_remaining_rule
endif
```

where one of two user defined rules would be selected depending on the value of a system variable NPARTS.

- In PROMODEL in-built elemental ***actions*** are used for resource allocation (GET), assembling parts (JOIN), sending parts to a resource (SEND [n] [part] TO [location]), etc in the OPERATION or OUTPUT part sections of the ***routing*** module. Composite rules can be constructed using the IF-THEN rule which access and manipulate part attributes, and system and user defined variables. The format of the IF-THEN rule is:

IF [condition] THEN [actions].

The ***condition*** part of the rule can be used to test the contents, capacity of resource, or the status of a resource, the value of a user defined variable, the clock value, the value of a part attribute, etc, In addition it is possible to drop out to an external language (C) if the required logic cannot be programmed using the in-built language.

- In ARENA the rules can be defined using the SIMAN base modules, which conform to the SIMAN block or experiment statements; Alternatively the Advanced Manufacturing Template can be used, which supports the majority of manufacturing systems from jobshop, to flow shops, to assembly lines. Model logic can be based on pre-defined selection rules, for example, the following priority rules for the selection of a part from a queue are available:-cyclic (CYC), random (RAN), preferred order rule (POR), largest number busy (LNB), largest remaining capacity (LRC), smallest remaining capacity (SRC). These are elemental rules which can be used alone or combined to form more complex composite rules.
- In FACTOR/AIM the rules are in-built and include:

1. AGV assignment rules provided are:- closest vehicle, closest cruising vehicle, closest parked vehicle, or longest idle vehicle.

2. Sequencing by:

   - Queue discipline (FIFO, LIFO).
   - Integer values (Attributes, Priority, Load-size, Number of Jobsteps).
   - Date/Time (Date, Jobstep Time, Processing Time Remaining).
   - Least slack-Static or Dynamic.

3. Order Release Rules:

   - Earliest Due Date.
   - Number of Jobsteps.
   - Order size.
   - Processing Time.

- Priority.
- Least Static Slack/Number of Jobsteps.
- Least Static Slack/Processing Time.

4. Material Handling Device contention rules based on:

- Earliest Due Date.
- Earliest Order Release Date.
- FIFO and LIFO.
- Attributes.
- Priority.
- Largest or Smallest Load size.

From the analysis of the behavioural elements of the generic simulators, it is evident that FACTOR/AIM is the easiest to use since no programming is required, but this is at the detriment of flexibility since only the in-built rules can be used with no scope for modification or user defined rules (Note:- The system, however, does allow the modeller to drop out into an external programming language to define rules that are not pre-defined within it; ability to define rules this way, however, is severely limited by the extent to which system status information can be communicated to and from the external program). The other manufacturing simulation systems are also similarly easy to use as long as only the elemental rules provided need to be used which, of course, would severely limit their application.

4.6 Common Modelling Elements of WITNESS, PROMODEL and FACTOR/AIM.

In this section the common elements, and their characteristics, of WITNESS, PROMODEL and FACTOR/AIM are identified and described using the classification given in Fig 4.6. Although manufacturing simulators are based on a common domain knowledge, their implementations differ in many respects, both in the way the model is specified and the way its data is stored. The three systems selected for a more in-depth study were chosen because of their differences; the core manufacturing activities (i.e. excluding transportation) are defined primarily through the real world objects such as parts and machines (locations in case of PROMODEL), but the emphasis placed on each is different.

4.6.1 Parts

A number of attributes of parts are commonly found in the majority of manufacturing simulators:

- The ***maximum arrivals*** of a particular part, which is the maximum number of parts which can be allowed during a single simulation run.
- The ***inter-arrival time*** is the time between successive arrivals.
- The arrival time of the first part of a part type, which is the simulation time at which the ***first part*** of a particular type ***arrives***.

- The ***lot size.***

These characteristics or attributes are specified in WITNESS, PROMODEL and FACTOR/AIM using fields within the the ***parts*** module (Fig 1 Appendix A), ***Part Scheduling*** Module (Fig 12 Appendix A) and ***demand*** editor (Fig 19 Appendix A) respectively.

4.6.2 The process plan

The process plan represents the path a part takes through the model including all processing steps and transportations. Consider the case, in Fig 4.7, where there are 3 machines and a load/unload station.

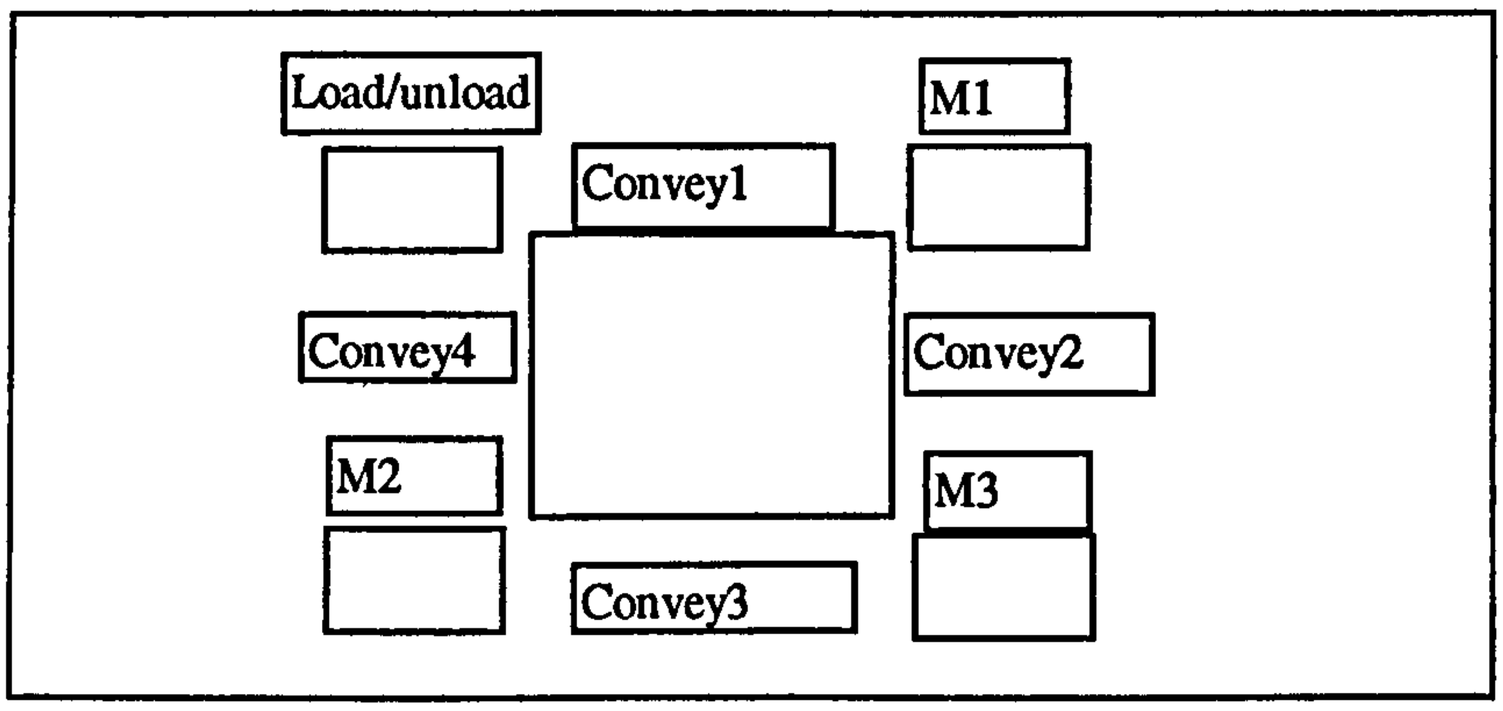


Fig 4.7 Three machine system

In WITNESS this path is entered using the ***route*** editor (Fig 3 Appendix A) within the ***part detail*** module. This editor allows the route to be specified using stages, where each stage represents an interaction with a transportation device or machine. The route stages for part 1 say, shown below, specify that it

visits machines m1, m2 and m3 using conveyors convey1, convey2, convey3 respectively, and the load/unload stages are 1 and 9 (Note:-This is an example of the internal representation of a process plan, and not the way the data is entered by the user).

```
PART ROUTE:  STAGE 1 :load/unload;
                R_SETUP : 0;
                R_CYCLE : 10;
        STAGE 2 : convey1@0;
                R_SETUP : 0;
                R_CYCLE : 0;
        STAGE 3 : m1;
                R_SETUP : 0;
                R_CYCLE : 7;
        STAGE 4 : convey2@0;
                R_SETUP : 0;
                R_CYCLE : 0;
        STAGE 5 : m2;
                R_SETUP : 0;
                R_CYCLE : 15;
        STAGE 6 : convey3@0;
                R_SETUP : 0;
                R_CYCLE : 0;
        STAGE 7 : m3@0;
                R_SETUP : 0;
                R_CYCLE : 5;
        STAGE 8 : convey4@0;
                R_SETUP : 0;
                R_CYCLE : 0;
        STAGE 9 : load/unload;
                R_SETUP : 0;
                R_CYCLE : 15;
        STAGE 10 : SHIP;
                R_SETUP : 0;
                R_CYCLE : 0;
```

In PROMODEL this process plan would be modelled via the ***routing*** module as shown in Fig 4.8.

Part	Location	Operation(min)	Output part	Next location	Condition	Qty	Move time(min)
p1	load/unload	10	p1	m1	0	1	conveyor[8]
p1	m1	7	p1	m2	0	1	conveyor
p1	m2	15	p1	m3	0	1	conveyor
p1	m3	5	p1	load/unload	0	1	conveyor
p1	load/unload	15	p1	exit	0	1	conveyor

Fig 4.8 PROMODEL routing module

Similarly in FACTOR/AIM the route a part takes is modelled via the definition of a process plan comprising a number of jobsteps. The first jobstep would be specified in the ***FIRST*** field, of the ***process plan*** editor (Fig 20 Appendix A), and in the above situation would be load/unload. The remainder of the process plan is specified using a complex field which has four columns for specifying:

- The job step name.
- The type of jobstep: a transportation (***move-between)*** or processing operation (***setup/operation)***.
- A description of the job step.
- The next job step

For an equivalent representation of the WITNESS and PROMODEL process plans, shown above, this would result in the jobsteps shown in Fig 4.9.

[8] The conveyor system would be specified using the conveyor location and conveyor transfer logic tables(see section 4.6.5.1)

Name	Type	Description	Next
js1	setup/operation[9]	load/unload station	js2
js2	move-between[10]	take loads to m1	js3
js3	setup/operation	process at m1	js4
js4	move-between	take loads to m2	js5
js5	setup/operation	process at m2	js6
js6	move-between	take loads to m3	js7
js7	setup/operation	process at m3	js8
js8	move-between	take loads to load/unload	js9
js10	Add_to_material	exit	

Fig 4.9 FACTOR/AIM Process Plan

If a conveyor is chosen as the transportation mechanism then the type of transportation in the ***move-between*** jobstep editor (Fig 23 Appendix A) is chosen as a conveyor system. A list of already defined conveyor systems is then available and a selection is made. The ***begin*** and ***end*** points, which indicate where the parts are picked up and dropped off respectively, are also defined for the ***move-between*** jobstep.

---

[9] The processing time would be entered using the Setup/Operation editor(Appendix A Fig 21) in the operation time(10 in our case) field.

[10] The conveyor system using the *move-between* jobstep editor(Appendix A Fig 23). For a thorough explanation see section 4.6.5.1

If a vehicle-track system is used, the transporter control points signifying the beginning and end points of the ***move-between*** jobtep would have to be specified.

The processing time at each location is modelled using the ***operation/setup*** jobstep editor by entering a value in the operation time field.

4.6.3 Buffers

There are two types of buffers commonly available as modelling elements in manufacturing simulators.

Machine dedicated buffers

A machine dedicated buffer is a queue at a machine. In a WITNESS model, it is represented through the ***machine's details*** (Fig 5 Appendix A). In PROMODEL, the buffers are defined as separate components in the ***capacities*** module before they are used in the ***routing*** module. For example the input/output buffers b1in and b1out of machine m1 would be represented, in the routing module, as shown in Fig 4.10.

Part	Location	Operation(min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	lo/unlo	0	p1	b1in	0	1	1
p1	b1in	7	p1	m1	0	1	conveyor
p1	m1	7	p1	b1out	0	1	conveyor
p1	b1out	5	p1	lo/unlo	0	1	conveyor
p1	lo/unlo	0	p1	exit	0	1	1

Fig 4.10 PROMODEL Representation of machine dedicated buffers in the routing module

In FACTOR/AIM machine dedicated buffers are modelled as ***general pools*** (Fig 24 Appendix A) and then assigned to a machine, using the ***setup/operation jobstep*** editor (Fig 21 Appendix A), in the Resource/Pool group. For example if B1out is the output buffer for m1 then it is specified in the Resource/Group field as shown in Fig 4.11

Name	Action
m1	Allocate
B1out	Free-After

Fig 4.11 FACTOR/AIM Output buffer representation in the Resource/Group field.

General Buffers

These buffers are not buffers used for queuing at resources, but are separate and used as temporary WIP storage. Two attributes are used to define their characteristics.

The **capacity** of the buffer is the maximum number of parts a buffer can hold at any particular time. In WITNESS the capacity of a buffer is entered directly into the ***capacity*** field within the ***buffer's details*** module (Fig 4 Appendix A), whereas in a PROMODEL simulation model the capacity, for say buffer b1in, is defined within the ***capacities*** module (Fig 13 Appendix A) in the ***Qty*** column. In FACTOR/AIM, buffers remote from machines are modelled as general pools with the capacity specified by using the ***general pool editor*** (Fig 24 Appendix A), in the capacity field; this is different from the dedicated buffers in that in the process plan the jobstep after the visit to the buffer will be **move_between**, signifying that the buffer is not attached to the machine.

Buffer Delay is the time elapsed before a part is allowed to leave a buffer. In WITNESS the delay is entered directly into the appropriate field in the ***buffers*** module (Fig 4 Appendix A), whereas in a PROMODEL simulation a delay of 5 time units of part p1 in buffer b1in, before visiting say machine m3, will be defined in the ***routing*** module in the ***operation*** column as shown in Fig 4.12

Part	Location	Operation(min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	b1in	5	p1	m3	0	1	conveyor

Fig 4.12 Specifying the delay of a part in a buffer in PROMODEL

In FACTOR/AIM the delay is modelled in the jobstep corresponding to the visit to the buffer by entering a value in the ***operation time*** field of the ***operation/setup*** jobstep editor (Fig 21 Appendix A).

4.6.4 Machines

Generic simulators usually allow four types of machines to be defined. These common types are:

1. A **single machine** that processes one part at a time. In WITNESS this type of machine is modelled by selecting the **'single'** option within the ***type*** field in the ***machine detail*** form (Fig 5 Appendix A), whereas in PROMODEL a single machine m1 is modelled by a simple routing entry in the ***routing*** module as shown in Fig 4.13.

Part	Location	Operation(min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	m1	5	p1	m1lout	0	1	conveyor

Fig 4.13 The PROMODEL representation of a single machine within the routing module.

In FACTOR/AIM a single machine is modelled by adding a ***single capacity*** resource (see Fig 25 Appendix A for resource editor), where all the processing characteristics are given via the process plan using the ***setup/operation jobstep*** editor (Fig 21 Appendix A).

2. A **batch machine** that processes a batch of parts at one time. In WITNESS a batch machine is modelled by:

   a) selecting the ***'batch'*** option in the machine detail (Fig 5 Appendix A);

   b) entering values for the ***minimum*** and ***maximum*** batch sizes that can be processed by the machine.

In PROMODEL[11] a batch machine m1 to process a batch of 3 units is modelled by specifying, in the ***routing*** module, the batch size in the ***quantity*** column of the location sending the part as shown in Fig 4.14.

---

[11]The capacity of m1 should be greater than the maximum batch size in the capacities module

Part	Location	Operation (min)	Output part	Next location	Condition	Qty	Move time(min)
p1	m3	5	p1	m1	0	3	conveyor
p1	m1	5	p1	m1	0	3	conveyor

Fig 4.14 The PROMODEL representation of a batch machine within the routing module.

In FACTOR/AIM a batch machine is modelled by defining a ***multi-capacity*** resource. The batch size is modelled using the ***accumulate jobstep*** (Fig 22 Appendix A) before the visit to the multi-capacity machine, where the size of the batch size is entered in ***Accum quantity*** field.

3. An **assembly machine** takes a number of parts and assembles them onto a single part. In WITNESS an assembly machine is modelled by:

   a) selecting the '***assembly***' option in the machine detail (Fig 5 Appendix A);

   b) entering a value for the number of parts to be assembled.

In PROMODEL a number of parts can be assembled into a single part using the ***JOIN*** action in the ***Output Part*** column of the machine sending the part, and in the ***Operation*** column of the assembly machine. The routing entries for modelling an assembly machine m3 which assembles parts p1 and p2 (quantity 3 for each) from machines m1 and m2 respectively onto a part called 'base' are shown in Fig 4.15.

Part	Location	Operation (min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	m1	5	p1	m3	JOIN	3	conveyor
p2	m2	5	p1	m3	JOIN	3	conveyor
base	m3	JOIN 3 p1	base	load/unload	0	1	conveyor
		JOIN 3 p2	base				

Fig 4.15 The PROMODEL representation of an assembly machine within the routing module.

In FACTOR/AIM an assembly machine is defined by specifying the jobstep that accesses it as an ***Accumulate/Split*** jobstep (Fig 22 Appendix A). The assembly quantity is specified by entering a value in the ***Accum quantity*** field.

4. A **production machine** takes in one part and outputs a number of parts.

   In WITNESS a production machine is modelled by:

   a) selecting the ***'production'*** type option in the machine detail (Fig 5 Appendix A).

   b) entering values for the ***quantity*** and ***name*** of the part to be produced.

In PROMODEL this is achieved by specifying the initial quantity, 1, in the ***quantity*** column of the machine sending the part and a ***quantity*** equal to the number produced in the ***quantity*** column of the production machine. The routing entries are given in Fig 4.16.

Part	Location	Operation (min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	m1	5	p1	m2	0	1	conveyor
p1	m2	5	p2	load/unload	0	10	conveyor

Fig 4.16 The PROMODEL representation of a production machine within the routing module.

In FACTOR/AIM a production machine is modelled in the same manner as an assembly machine. This again requires the ***jobstep*** which accesses the production machine being specified as ***Accumulate/Split*** (Fig 22 Appendix A), but this time a production quantity is entered in the ***split*** field.

Machine setups

There are usually two types of setups:- one after a change of part type and the other after a number of operations have been completed. It was found that the type of setup most commonly available for modelling in generic simulators is the one after a ***change of part type***, although some generic simulators like WITNESS support both types.

In WITNESS to achieve the change of part type setup the setup mode is set to ***after a part change*** (Fig 5 Appendix A). and the ***setup time*** specified,

whereas in a PROMODEL simulation a setup after a part change is specified in the ***Downtime*** module as shown in Fig 4.17.

Basis	Resource	Part for which setup occurs	Duration (Minutes)	Preceding Part	Qty	Maintenance Resource
Setup	m1	All	10	All	1	man1

Fig 4.17 Downtimes Module

In a FACTOR/AIM model a setup after a part change is specified in the ***setup/operation*** jobstep editor in ***setup time*** field (Fig 21 Appendix A).

Machine Breakdowns

In generic manufacturing simulators the following types of breakdowns are usually modelled:

1. Breakdown according to ***Available time***. In WITNESS if there is a ***breakdown*** to be modelled in terms of the time the machine was available then:

   a) select the ***available time*** option in the ***breakdown*** field of the ***machines*** module (Fig 5 Appendix A).

   b) enter time between breakdowns.

   c) enter repair time.(Note:-in all cases, time can be defined by a constant, an expression or a statistical distribution).

For PROMODEL a **breakdown** in terms of the time the machine was available is modelled by:

a) setting the ***Basis column*** to **Clock** in the ***Downtime*** module (Fig 14 Appendix A).

b) entering a time between breakdowns.

c) entering a repair time.

In FACTOR/AIM a number of different breakdowns can be defined via the ***breakdown editor*** (Fig 26 Appendix A) and then assigned to particular machines. In order to specify a breakdown according to available time:

a) select the breakdowns value basis as ***Onshift Time.***

b) enter time between breakdowns.

c) enter repair time.

2. Breakdown after a **number of operations.** In WITNESS if breakdown after a number of operations is to be modelled for a particular machine, then:

   a) select ***number of operations*** option in the ***breakdown*** field of the machines module (Fig 5 Appendix A).

   b) enter number of operations between breakdown.

   c) enter repair time.

In PROMODEL a breakdown after a number of operations have been completed is achieved by:

a) setting the entry in the ***Basis column*** to ***Cycle*** in the downtime module (Fig 14 Appendix A); the resource is removed from service after the specified number of operation cycles.

b) entering the number of operations between breakdown.

c) entering the repair time.

In FACTOR/AIM to model a breakdown after a number of operations then:

a) set ***count*** as ***value basis*** in the ***breakdown editor*** (Fig 26 Appendix A)

b) enter number of operations

c) enter repair time.

3. Breakdown according to the time the **machine** is **busy.** In WITNESS if a **breakdown** in terms of the time the **machine** is **busy** is to be modelled for a particular machine then:

a) select the ***busy time*** option in the ***breakdown*** field of the ***machines*** module (Fig 5 Appendix A).

b) enter interval between breakdowns.

c) enter repair time.

In PROMODEL a breakdown in terms of the time the machine is busy is modelled by

a) setting the entry in the ***Basis column*** to ***Usage*** in the ***Downtime*** module.

b) entering interval between breakdowns.

c) entering repair time.

It should be noted that in WITNESS the labour for repair is not detailed within the breakdown field, but is specified by:

a) selecting the ***Repair*** option within the ***Labour*** field.

b) specifying a labour rule for acquiring labour of a particular type using the ***labour editor.***

This is not the case with PROMODEL, where the labour name required for repair is specified within the ***maintenance resource*** column of the ***downtimes*** module (Fig 14 Appendix A).

In FACTOR/AIM to model a breakdown by the time the machine is busy:

a) set ***value basis*** in the ***breakdown*** editor (Fig 26 Appendix A) to ***processing time.***

b) enter time between breakdowns.

c) enter the repair time.

4.6.5 Conveyors

Conveyors along with vehicle and track systems are the main transportation devices that can be modelled in generic simulators. In WITNESS a number of conveyors are usually linked together to form a conveyor system. In PROMODEL and FACTOR/AIM, however a conveyor system is modelled as a combination of conveyor sections, where a conveyor section is defined as any uninterrupted span of conveyor of the same type.

In WITNESS a conveyor system is modelled as a number of different inter-connected conveyors via the ***conveyors detail*** module.

The conveyor system in PROMODEL is modelled via the Conveyor module, which comprises of the following three sub-modules: ***Conveyors Specification***, the ***Conveyor Location Interface*** and the ***Conveyor Transfer Logic*** sub-modules.

In FACTOR/AIM the conveyor system comprises a number of ***conveyor control points linked into sections***. For example, a conveyor section cs1 could link conveyor control points cp1 and cp2. There are three attributes associated with conveyors:-its type, capacity and cycle time; in addition, how the parts are routed with a conveyor need to be defined.

There are usually two different ***types of conveyors*** that can be modelled. In WITNESS the two types are modelled by entering either ***fixed or queuing*** in the ***type field*** of the ***conveyor detail module*** (Fig 6 Appendix A). In PROMODEL there are also two types which can be modelled and are compatible with the WITNESS classification of a conveyor; a *T* (fixed) entry in the ***Type column*** of the conveyor specifications sub-module (Fig 15 Appendix A) indicates a ***transport conveyor*** and a *A* (queuing) entry an ***accumulation conveyor***. Examples of fixed conveyors are belt, chain, tray and trolley, while examples of queuing conveyors are roller and towline. In FACTOR/AIM the type of the conveyor is specified using ***the conveyor system editor*** (Fig 27 Appendix A); for a queuing conveyor the ***block rule*** is selected as ***Accumulate,*** and for a fixed conveyor the ***block rule*** is selected as ***Block.***

In WITNESS the ***capacity*** of a conveyor is specified in the ***part length*** field of the ***conveyor*** module (Fig 6 Appendix A) which signifies the maximum number of parts that can be physically placed end to end on the conveyor. In PROMODEL the capacity of each section is modelled by entries for the ***length of load*** and ***spacing column***s of the ***conveyor specifications*** module (Fig 15 Appendix A). In FACTOR/AIM capacity relates to that of a particular conveyor section and is specified in the ***capacity*** field using the ***conveyor segment*** editor (Fig 28 Appendix A).

The ***cycle time*** is the time to move a part through one part length. In WITNESS this is modelled by using the ***cycle time*** entry in the ***conveyor***

detail module (Fig 6 Appendix A), whilst in PROMODEL the cycle time is modelled by entering it in the ***speed*** column of the ***conveyor specifications*** sub-module (Fig 15 Appendix A). In FACTOR/AIM the cycle time is specified using the ***conveyor system*** editor (Fig 27 Appendix A) in the ***velocity*** field.

4.6.5.1 How parts are routed using a conveyor.

This is used to indicate how the conveyor system is connected to the load/unload stations and machining centres. In WITNESS this is specified by the input/output rules and entry of the route in the part's details, which specifies which conveyor(s) are used to transport a part to a specific location. The input rule is set to **WAIT** and the output rule to **PUSH TO ROUTE**.

Consider the layout of Fig 4.18, a 4 station system served by 4 conveyors.

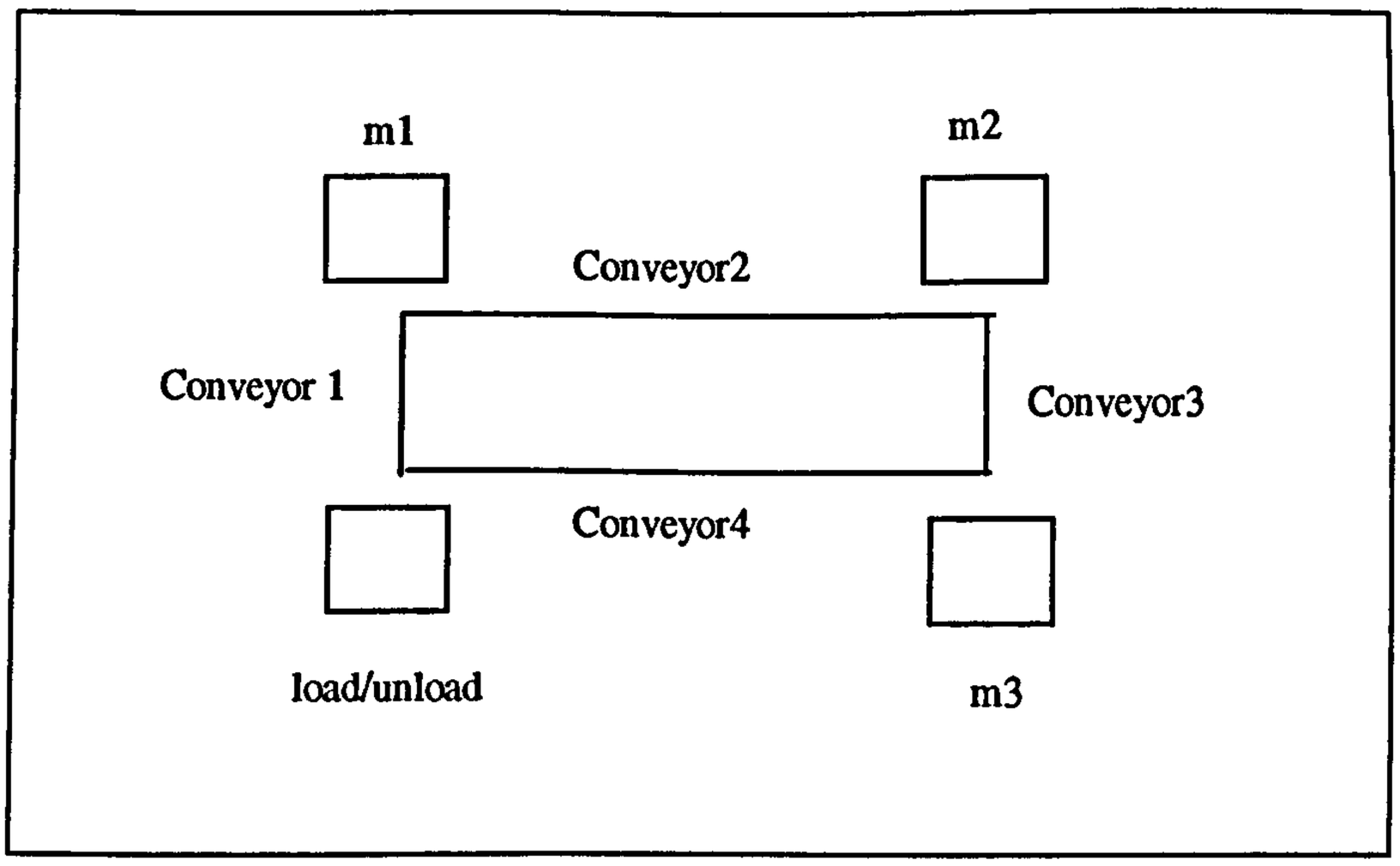


Fig 4.18 A WITNESS representation of a 4 station and 4 conveyor system

We see that machine m1 is connected to conveyor1, machine m2 to conveyor2, machine m3 to conveyor3 and the load/unload to conveyor4. These connections would be specified in the ***route entry*** in the detail of part p1 as:

```
PART ROUTE:        STAGE 1 :load/unload;
                        R_SETUP : 5;
                        R_CYCLE : 10;
                   STAGE 2 : conveyor1@0;
                        R_SETUP : 0;
                        R_CYCLE : 0;
                   STAGE 3 : m1;
                        R_SETUP : 0;
                        R_CYCLE : 5;
                   STAGE 4 :conveyor2;
                        R_SETUP : 0;
                        R_CYCLE : 15;
                   STAGE 5 : m2@0;
                        R_SETUP : 0;
                        R_CYCLE : 0;
                   STAGE 6 : convey3@0;
                        R_SETUP : 0;
                        R_CYCLE :
                   STAGE 7 : m3@0;
```

R_SETUP : 0;
R_CYCLE : 0;
STAGE 8 : convey4@0;
R_SETUP : 0;
R_CYCLE : 0;
STAGE 9 : load/unload;
R_SETUP : 0;
R_CYCLE : 15;
STAGE 10 : SHIP;
R_SETUP : 0;
R_CYCLE : 0;

This 4 conveyor system would be represented in PROMODEL as a single conveyor comprising 4 sections as shown in Fig 4.19.

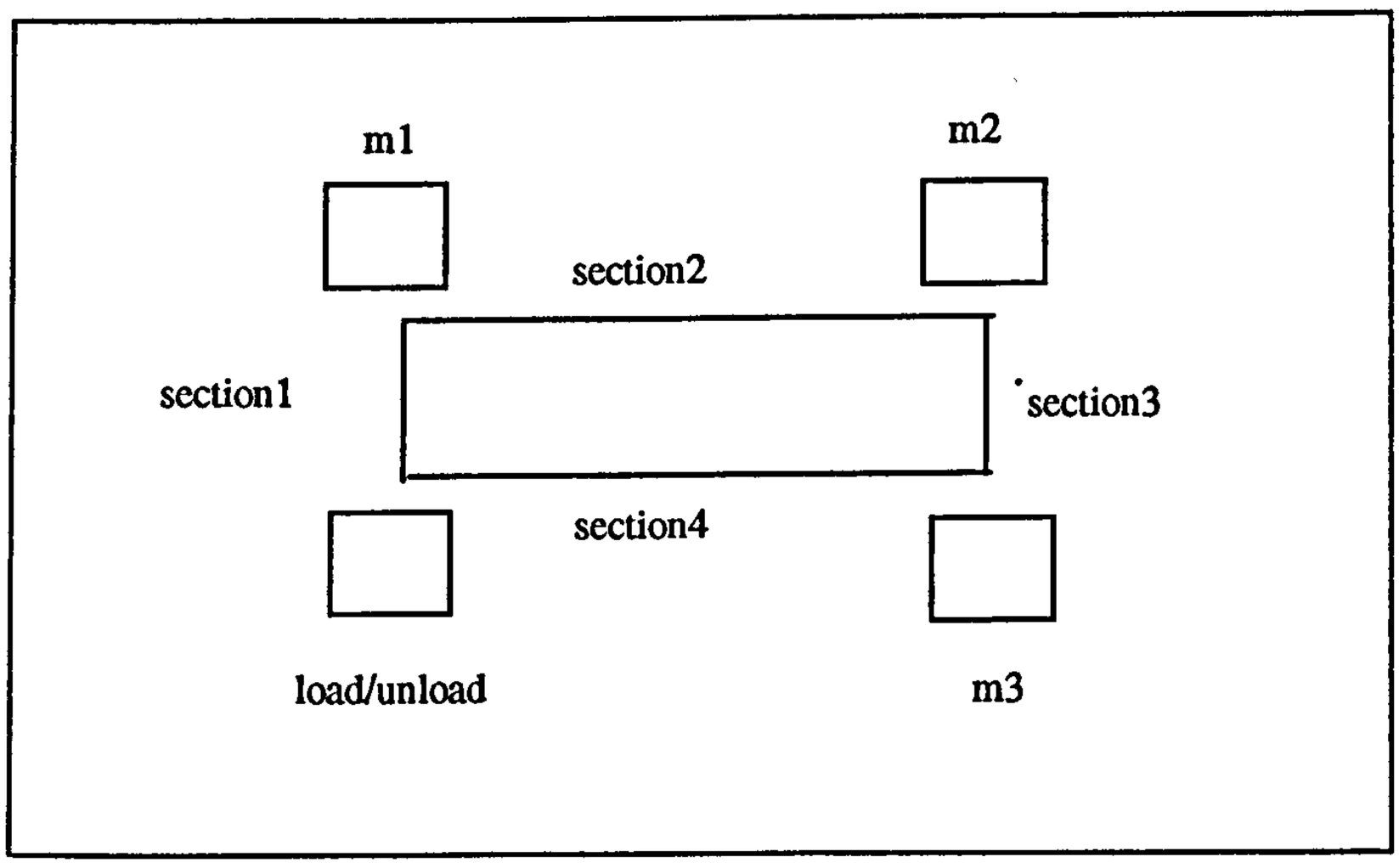


Fig 4.19 PROMODEL representation of 4 conveyor system

In PROMODEL these connections would be specified in ***the Conveyors Location Interfaces*** sub-module as shown in Fig 4.20.

CONVEYOR LOCATIONS INTERFACE

Location	Conv	Ft(m)	Sec
m1	section1	1	1
m2	section2	1	1
m3	section3	1	1
load/unload	section4	1	1

Fig 4.20 Conveyor Location interface entries for PROMODEL example

The ***Ft(m)*** entry is the distance along the section at which the routing location interfaces with the conveyor and the ***sec*** entry is the time to transfer between the conveyor and the location (Note:-***sec*** stands for seconds but, in practice, the time can be specified in any unit).

In PROMODEL the ***Conveyor Transfer Logic*** sub-module is used to specify the length of the conveyors and their connections to other conveyors. The layout of the Conveyor transfer logic is shown in Fig 4.21.

CONVEYOR TRANSFER LOGIC

From	Position ft(m)	To	Position ft(m)	Time (Sec)
c1	3	c2	0	1
c2	4	c3	0	1
c3	3	c4	0	1
c4	4	c1	0	1

Fig 4.21 Conveyor Transfere logic entries for PROMODEL example

- **From** is any conveyor section which merges into another conveyor section.

- **Position** is the distance of the load along the "**From**" conveyor where transfer takes place (conveyor length).
- **To** is any conveyor section which the "**From**" section merges into when loads are transferred.
- **Position** is the distance of the load along the "**To**" conveyor after transfer has taken place (0 to signify beginning of conveyor section).

FACTOR/AIM uses a similar method to PROMODEL for specifying a conveyor with the use of conveyor control points connected by a number of conveyor sections. The 4 conveyor system would be represented using conveyor control points ccp1, ccp2, ccp3 and cpp4, connected by 4 sections section1, section2, section3 and section4. This is shown in Fig 4.22.

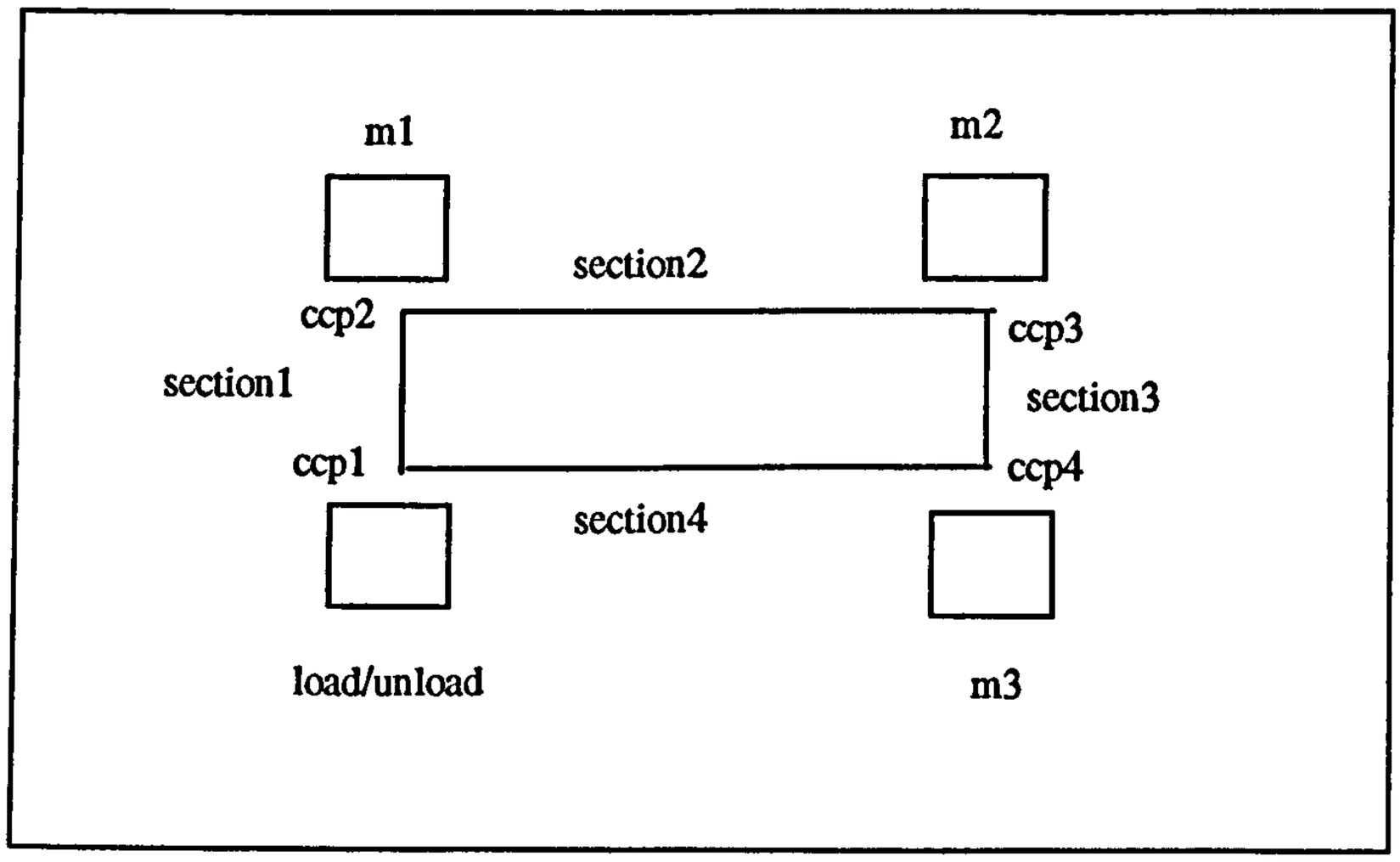


Fig 4.22 FACTOR/AIM representation of a 4 station and 4 conveyor system

The connections to resources would be specified using the ***move-between*** jobstep editor (Fig 23 Appendix A), of the process plan in the ***origin***, ***destination*** and ***drop off*** fields. This is illustrated in Fig 4.23, where:

- **Origin** is the conveyor control point from which the part is transported.
- **Destination** is the conveyor control point to which the part is transported.
- **Drop off** is the location which interfaces with the destination control point.

Job step	Origin	Destination	Drop off
move part from load/unload to m1	ccp1	ccp2	m1
move part from m1 to m2	ccp2	ccp3	m2
move part from m2 to m3	ccp3	ccp4	m3
move part from m3 to load/unload	ccp4	ccp1	load/unload

Fig 4.23 FACTOR/AIM jobsteps for specifying conveyor section connections to resources

4.6.6 Transporter Systems

A transporter system in generic manufacturing simulators is defined using a fixed path along which a vehicle moves; both the vehicle element and the track system on which it travels have to be defined together with their attributes and characteristics. In addition, the way vehicles search for work and are routed through the tracks have to be specified.

4.6.6.1 Vehicles

The common characteristics that can be associated with a vehicle are: the maximum number of parts a vehicle can carry (capacity), its start position, and the time to pickup and deposit parts.

The ***capacity*** of a vehicle is modelled by using the ***max parts*** field of the WITNESS ***vehicle detail*** module (Fig 7 Appendix A). In PROMODEL the number a vehicle can transport is specified in the ***routing*** module as an entry in the ***Qty*** column. Similarly in FACTOR/AIM the maximum number of parts a vehicle can carry is specified using the ***load*** button within ***the transporter vehicle*** editor (Fig 31 Appendix A).

The ***start position*** of a transporter is the track where the vehicles enter the simulation. In WITNESS the start position is entered in the ***vehicle detail*** module (Fig 7 Appendix A) by entering an ***output rule*** specifying the track to which vehicle is initially ***pushed to*** when the simulation starts e.g. PUSH to TRACK1. In PROMODEL this is entered as a path point directly into the ***start position*** column of the ***transporter specifications*** sub-module (Fig 18 Appendix A) In FACTOR/AIM the start position of the transporter is specified in the ***initial position*** field of the ***transporter vehicle*** editor (Fig 20 Appendix A).

The ***pick up*** time is the time taken to load a part onto the vehicle. In WITNESS this is specified in the ***track detail*** form (Fig 8 Appendix A) as an entry in the ***load*** detail, whilst in PROMODEL it is specified in the ***transporter specification*** module (Fig 18 Appendix A) as an entry in the ***pickup time*** column. In FACTOR/AIM the pick up time is specified in the ***pickup time*** field of the ***transporter vehicle*** editor (Fig 31 Appendix A).

The deposit time is the time taken to unload a part from a vehicle. In WITNESS this is specified in the ***track detail form*** (Fig 8 Appendix A) as an entry in the ***unload*** detail, whilst in PROMODEL it is specified in the ***transporter specification*** module (Fig 18 Appendix A) as entry in the ***deposit time*** column. In FACTOR/AIM the deposit time is specified in the ***deposit time field*** of the ***transporter vehicle*** editor (Fig 31 Appendix A).

4.6.6.2 Tracks

Tracks are the paths on which the vehicles travel, delivering parts between various processing stations. The different generic manufacturing simulators have distinct but equivalent methods for representation of tracks. For Example, a system comprising 5 machines, a load/unload station and 6 tracks would have representation in WITNESS shown in Fig 4.24.

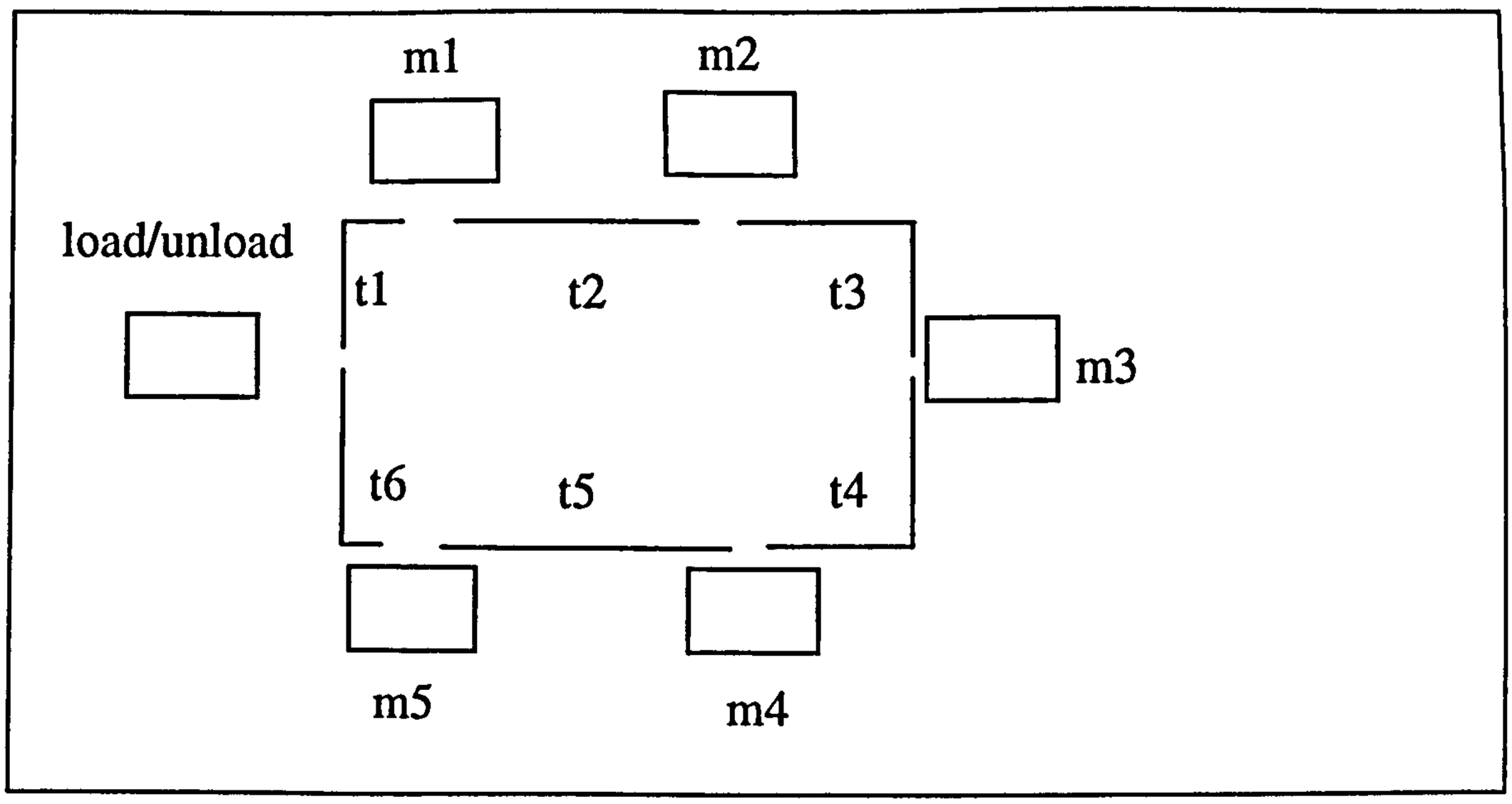


Fig 4.24 WITNESS Representation of a 6 machine/6 track system

The equivalent PROMODEL representation is shown in Fig 4.25 where p1 to p2, p2 to p3, p3 to p4, p4 to p5, p5 to p6, p6 to p1 are the equivalent path point connections of the WITNESS tracks t1,...,t6.

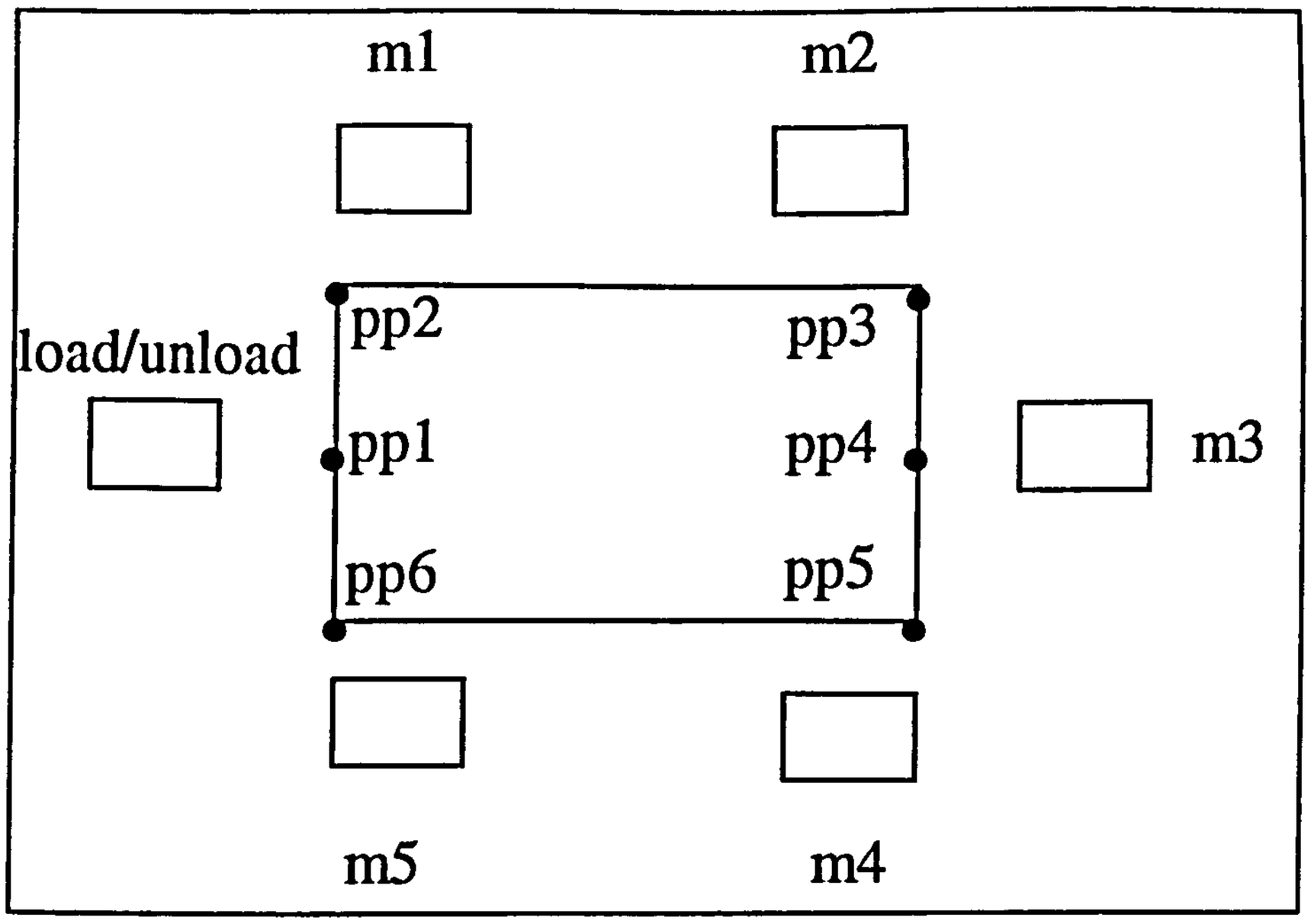


Fig 4.25 PROMODEL 6 machine/6 track system representation

The FACTOR/AIM representation, shown in Fig 4.26, is similar to the PROMODEL one with transporter control points tcp, ... ,tcp6 connected by transporter segments tseg1,....,tseg6.

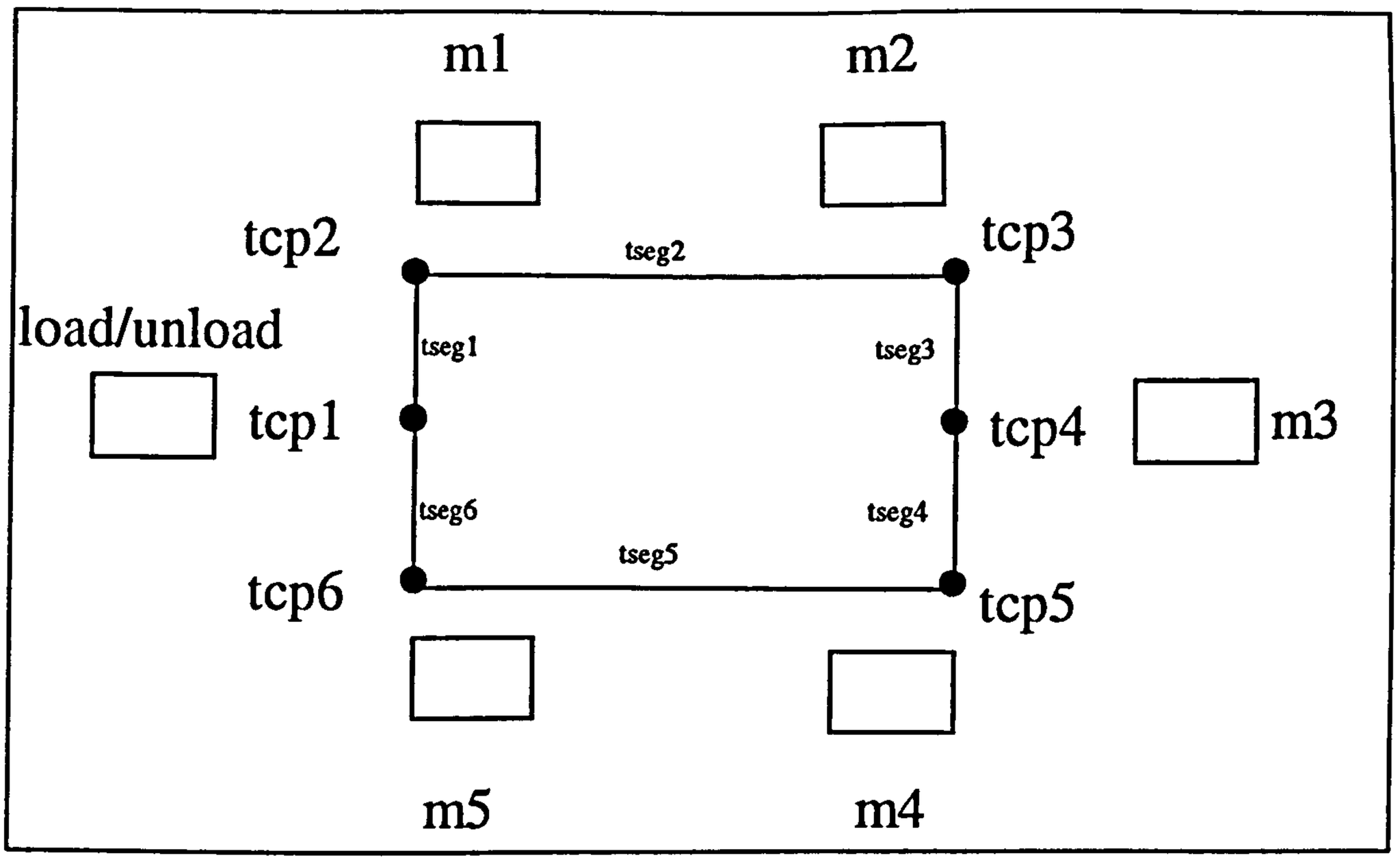


Fig 4.26 FACTOR/AIM 6 machine/6 track system representation

Tracks define the designated guided path along which vehicles can travel. The characteristics commonly used to specify tracks are:-the way they are connected, their length and maximum speed.

Track connections are used to specify which track a vehicle enters once it reaches the end of the track it is traveling on at present. In WITNESS this is specified as ***an output rule*** for a track in the ***track detail*** form (Fig 8 Appendix A). In our example the output rule for:

- track t1 is PUSH TO t2
- track t2 is PUSH TO t3
- track t3 is PUSH TO t4, etc

In PROMODEL the same connections are specified in the ***transporter path logic*** sub-module as shown in Fig 4.27.

PATH LOGIC FOR(v1)

From	To	Block	Speed ft(m)/min	Distance ft(m) or time(min)
pp1	pp2			
pp2	pp3			
pp3	pp4			
pp4	pp5			
pp5	pp6			
pp6	pp1			

Fig 4.27 Path logic for PROMODEL example

In FACTOR/AIM the track segment connections are specified in terms of the transporter control points via the use of the ***transporter segment*** editor (Fig 33 Appendix A). In the example shown, the segments and their ***begin*** and ***end*** points would result in the entries shown in Fig 4.28.

Segment	Begin	End
tseg1	tcp1	tcp2
tseg2	tcp2	tcp3
tseg3	tcp3	tcp4
tseg4	tcp4	tcp5
tseg5	tcp5	tcp6
tseg6	tcp6	tcp1

Fig 4.28 Segment information for FACTOR/AIM model

In WITNESS the ***length*** of the track is used to calculate the vehicle movement times along the track, and is modelled by an entry in the ***length*** field in the tracks detail form (Fig 8 Appendix A). In PROMODEL tracks along which the vehicles travel are defined in the ***Transporter Path Logic*** sub-module as path point connections which make up the path logic for a transporter; transporters can only move from path point to path point, and the distance between them are defined. For example track tr1 (length 10m) which is connected to track tr2 (length 5m) could be defined as connect path points pp1, pp2 and p3. This is shown in Fig 4.29.

PATH LOGIC FOR(v1)

From	To	Block	Speed ft(m)/min	Distance ft(m) or time(min)
pp1 pp2	pp2 pp3			5 10

Fig 4.29 Path Logic for PROMODEL example

Similarly In FACTOR/AIM the length of a segment is specified via the ***transporter segment*** editor (Fig 33 Appendix A).

The ***maximum speed*** a vehicle is allowed to travel along the track is modelled in WITNESS by an entry in the ***max speed*** field in the ***tracks*** detail (Fig 8 Appendix A) form, whilst in PROMODEL this is defined in the ***Transporter Path Logic*** sub-module as an entry in the ***speed*** column of the path point connections. In FACTOR/AIM the only way of specifying a maximum speed along a track is to enter a velocity for the vehicle in the ***velocity*** column of the ***transporter fleet*** editor (Fig 29 Appendix A).

4.6.6.3 Work search

The way a vehicle searches for work in WITNESS is by the use of a work search list, specified for all tracks using the ***work search*** option within the ***track*** detail.(Appendix A Fig 8). This is a list of tracks where parts may be waiting to be loaded. Typically, this list includes the tracks which are a short

distance from the current track. Any number of tracks may be specified in this list. When an idle vehicle reaches the front of a track, it scans down its demand list. If it encounters a demand to load at a track which appears in the work search list of the current track, it will be assigned to the CALL. If the load track is the current track, the vehicle immediately attempts to load. The order of the work search list does not determine the order in which demands are satisfied. For example, the work search list for a track may be: T1, T2. However, if the first entry in the demand list requires loading at T2, this may be satisfied before T1. This method requires all load tracks to have a calling mode of CALL; for example CALL AGV1, TI, T2 means that transportation is required using AGV1 from tracks T1 to T2.

In PROMODEL this type of work search is achieved using the ***Interface*** and ***search priority*** sub-modules, where the tracks tr2, tr3 and tr4 would be represented by path points pp2 to pp3, pp3 to pp4, and pp4 to pp5 respectively. These path points are assigned locations corresponding to loading/unloading points via the ***Location Interface*** sub-module, which identifies the connections between the ***Locations (machines)*** and the ***Location Interface points (Path points).*** In our example this would result in the entries shown in Fig 4.30, where:

- Location is the part routing location with which the transporter interfaces.
- Point is the transporter point where the transporter interfaces with the routing location.

LOCATION INTERFACES

Location	Point
load/unload	pp1
m1	pp2
m2	pp3
m3	pp4
m4	pp5
m5	pp6

Fig 4.30 Location Interfaces for PROMODEL example

In the ***Search priority*** module ***Type*** is the type of search priority and would be ***W for work.*** If a sequence is specified for this type then the transporter searches for work at each routing location. If no locations are specified or if no parts are waiting at any of the specified locations, then the transporters work search will default to the ***closest waiting part*** rule. In our example this would result in the entries shown in Fig 4.31. This means if there is no work at path point pp2 then search machines m2, m3 and m4, which correspond to transporter sections ending at transporter path points pp3, pp4 and pp5 respectively.

SEARCH PRIORITY

Type	Point	Sequence
W . .	pp2	m2,m3,m4, END

Fig 4.31 Search Priority for PROMODEL example

In FACTOR/AIM once a part has been processed and if the ***move_between*** jobstep (Fig 23 Appendix A)) indicates a transporter vehicle is required, a request for the vehicle is logged. The assignment of the vehicle is then to the part that is closest to the vehicle.

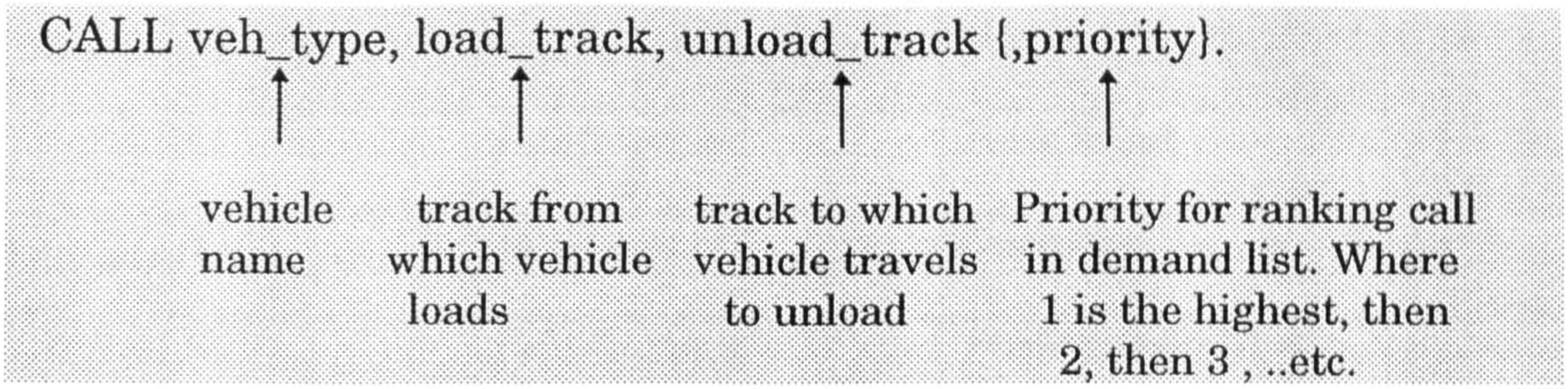
4.6.6.4 Part routing

Suppose the machine visitation order of part p1 is load/unload, m3, m5, m1 and load/unload for the WITNESS representation of Fig 4.24, then load and unload tracks that correspond to the machine visits for part p1 in the table is as given in Fig 4.32.

To achieve visit to station	Load track	Unload track
m3	t1	t3
m5	t4	t5
m1	t6	t1
load/unload	t2	t6

Fig 4.32 Load/Unload tracks for WITNESS example

A CALL action statement is used in the ***finish cycle actions*** for a machine (Fig 5 Appendix A), where WITNESS keeps a list of unsatisfied calls for each type of vehicle called the **DEMAND LIST**. The syntax of the CALL statement is:



The CALL statements for the ***actions on finish*** after processing at machines m3, m5 and m1 are given in Fig 4.33.

Machine on which processing complete	Call statement in actions on finish
m3	CALL vehicle1,t1,t3.
m5	CALL vehicle1,t4,t5.
m1	CALL vehicle1,t6,t1.

Fig 4.33 Call statements in actions on machine finish cycle for WITNESS example

The connections of the tracks to machines would be indicated by specifying load/unload rules for the tracks. The machine-track connections shown in the representation in Fig 4.24 would require the track load/unload rules shown in Fig 4.34.

Track	Load rule	Unload rule
t1	Pull from m1	Push to m1
t2	Pull from m2	Push to m2
t3	Pull from m3	Push to m3
t4	Pull from m4	Push to m4
t5	Pull from m5	Push to m5
t6	Pull from load/unload	Push to load/unload

Fig 4.34 Load/Unload rules for WITNESS example

In PROMODEL this visitation order would be represented in the ***routing*** module, as shown in Fig 4.35.

Part	Location	Operation (min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	load/unload	0	p1	m3	0	1	agv
p1	m3	4	p1	m5	0	1	agv
p1	m5	4	p1	m1	0	1	agv
p1	m1	3	p1	load/unload	0	1	agv
p1	load/unload	0	p1	exit	0	1	agv

Fig 4.35 Transporter routing in PROMODEL

The routing in a FACTOR/AIM model incorporating a transporter would be specified via the process plan. The transporter control point connections to resources would be specified using the ***move_between*** jobsteps (Fig 23

Appendix A), of the process plan in the ***origin, destination*** and ***drop off*** fields. This is illustrated in Fig 4.36 for the layout shown in Fig 4.26,. where:

- **Origin** is the conveyor control point from which the part is transported.
- **Destination** is the conveyor control point to which the part is transported.
- **Drop off** is the location which interfaces with the destination control point.

Job step	Origin	Destination	Drop off
move part from load/unload to m1	tcp1	tcp2	m1
move part from m1 to m2	tcp2	tcp3	m2
move part from m2 to m3	tcp3	tcp4	m3
move part from m3 to m4	tcp4	tcp5	m4
move part from m4 to m5	tcp5	tcp6	m5
move part from m5 to load/unload	tcp6	tcp1	load/unload

Fig 4.36 Transporter control point machine connections for FACTOR/AIM example

4.6.7 Labour

Labour is the manpower required for the operation, setup or repair of a machine and can be shared by several machines. Two characteristics of labour are commonly defined:-its quantity and its assignment.

In WITNESS the ***quantity*** is the number of a labour type which have the same operating characteristics and is specified in the ***quantity*** field of the

labour detail (Fig 11 Appendix A), whereas in PROMODEL the number of a particular labour type is entered in the ***capacities*** module (Fig 13 Appendix A) in the ***quantity*** column. In FACTOR/AIM labour is modelled as a multi-capacity operator, with the quantity specified via the ***multi-capacity operator editor*** (Fig 34 Appendix A) in the ***quantity*** field.

Once labour has been defined it must be ***assigned to*** resources. It is commonly assigned for:

1. **Operating a resource.** In WITNESS the labour for operating a machine is specified by activating the ***Cycle*** button (Fig 5 Appendix A) within the ***Labour*** field of the machine detail and entering a rule for acquiring a particular labour type. In a PROMODEL simulation model the labour requirement, of man1, for machine m1 to cycle part p1 would be defined in the ***routing*** module using the ***Get and Free actions*** within the Operation column as shown in Fig 4.37.

Part	Location	Operation(min)	Output part	Next location	Condi-tion	Qty	Move time(min)
p1	m1	Get man1 5 Free man1	p1	outbuff 1	0	1	conveyor
p1	outbuff1	0	p1	m3	0	1	conveyor

Fig 4.37 Labour assignment in PROMODEL

In FACTOR/AIM labour for operating a resource is assigned in the ***Setup/Operation*** jobsteps under the ***Resource/Pool/Group*** heading. For example if we wanted to assign multi-capacity operators called ***workers*** in the ***Resource/Pool/Group*** of the ***Setup/Operation jobstep*** (Fig 21 Appendix A) we would have the entries in the name and action columns shown in Fig 4.38.

Name	Action
workers	Allocate/Free

Fig 4.38 Labour assignment in FACTOR/AIM

2. **Setup**. In WITNESS the labour for setup is specified by activating the ***setup*** button within the ***Labour*** field of the machine detail (Fig 5 Appendix A) and entering a rule for acquiring a particular labour type, whereas in PROMODEL it is specified within the ***downtimes*** module in the ***maintenance resource*** column as shown in Fig 4.39.

Basis	Resource	Part for which setup occurs	Duration (Minutes)	Preceding Part	Qty	Maintenance Resource
Setup	m1	All	10	All	1	man1

Fig 4.39 Specification of labour for setups in PROMODEL

In FACTOR/AIM labour for setting up a resource is assigned in the ***Setup/Operation*** jobsteps (Fig 21 Appendix A) under the ***setup*** heading in the ***resource*** field.

4.7 Discussion

It has become evident from the analysis of a number of popular generic manufacturing simulators that they provide a number of common modelling elements. These elements, although they may have different names, serve the same modelling purpose. For example processing stations in WITNESS are defined using the Machine element, whilst in PROMODEL they are defined using the Location element.

The analysis has also shown that the main difference between the simulators are in their world views, internal data representations and the interfaces used by the user to specify the model. The world view is predominately activity scanning or process interaction based since most generic simulators modelling capabilities revolves around the definition of processing stations and materials respectively. There are, however, exceptions like WITNESS which are hybrid systems using a mixture of activity and process interaction world views.

A more in-depth analysis of WITNESS, PROMODEL and FACTOR/AIM has shown that although there are differences in the data models (consisting of the internal representation and the user interface used to map the manufacturing

systems) of the different simulators, the modelling elements and their characteristics used to represent the manufacturing system are, to a great extent, the same or equivalent in nature. However, some generic simulators allow the modelling of certain manufacturing component characteristics, whilst others don't; as an example WITNESS allows the modelling of conveyor breakdown and PROMODEL doesn't.

The identification and the classification of the common elements and their characteristics has shown that they can be used as a blue print for a framework that can be used in the development of future generic simulators; this can be achieved by using the classification as a minimum modelling requirement. It is also evident that the modelling elements that are common to the majority of simulators are those used to model the structural components (resources, facilities, etc. and their characteristics) of manufacturing systems. There, however, seems to be a lack of modelling conformity across the various generic simulators for the behavioral modelling elements (priority rules, resource allocation rules); some generic simulators have in-built rules (FACTOR/AIM) with no scope for modification or tailoring, while others (PROMODEL, WITNESS and ARENA) require the behavioural element to be developed using the in-built programming language. Also, when behavioural rules have to be programmed, different generic simulators use different approaches; for example, WITNESS uses INPUT/OUTPUT rules and ACTIONS within the detailing modules of the various resource elements for specification of behavioural rules, while PROMODEL uses rules only within

the OPERATION and OUPUT PART columns of the routing module to specify priorities of the jobs and their destinations when alternatives exist.

4.8 Development of a Standard Framework for Manufacturing Simulation.

The structure of a manufacturing simulator can be divided up into three main elements:- an internal representation of the manufacturing system model, a user interface for the user to populate the internal representation with the relevant data, and a simulation executive which executes the model; in addition, a simulator would contain a number of utility routines to support the data model and/or the executive and for generating reports on experimental results. The internal representation variously takes the form of a list file (WITNESS), tables (PROMODEL), databases (FACTOR/AIM), etc. In some cases, the user interface simply allows the user to populate directly a series of tables that constitute the internal representation. Others have menu driven forms which are filled in by the user to specify the model, and data is then mapped onto internal model representation through an internal translation process; the populated forms themselves may be stored as a model of the manufacturing system, but are not directly accessed by the executive. The overall structure of a generic manufacturing simulator is shown in Fig 4.40.

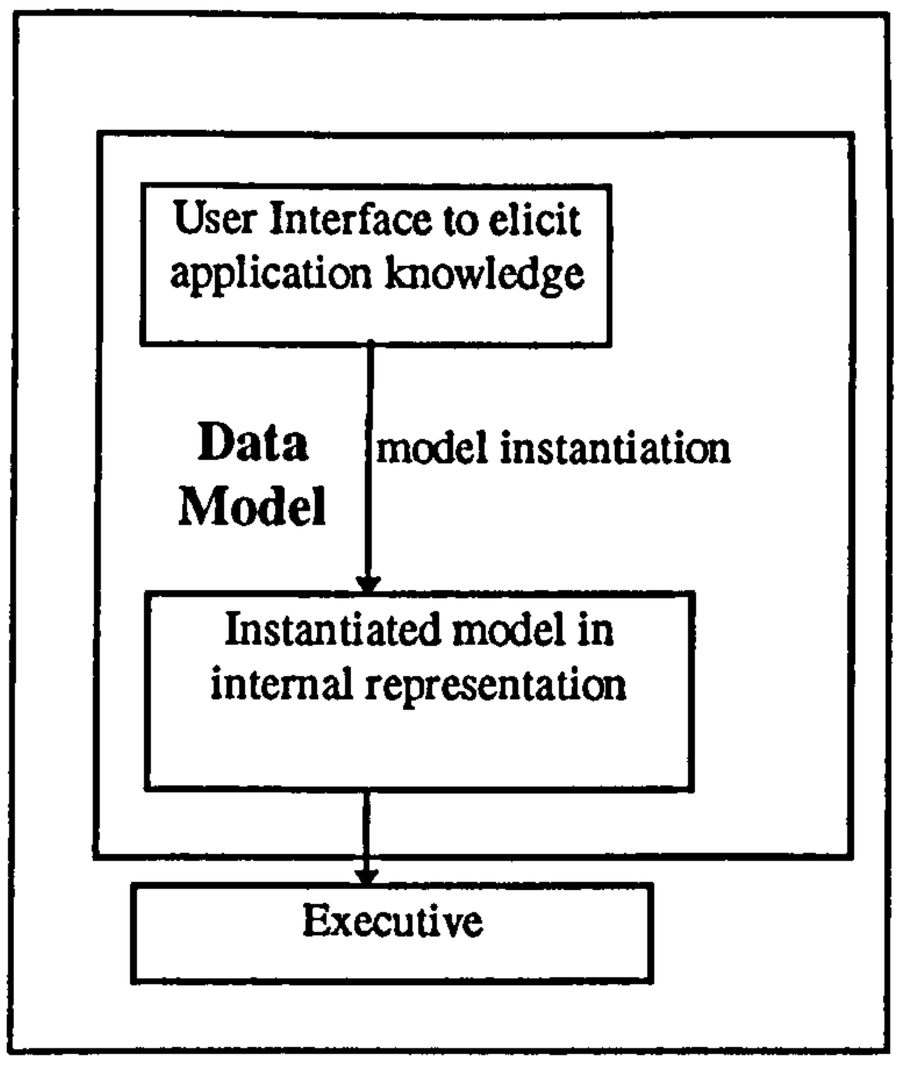


Fig 4.40 Overall Structure of Generic Manufacturing Simulator

The data models in the simulators are tightly coupled, and are very distinctive in the character of the user interface as well as the way the data is organised both for it and the internal representation. This makes it difficult for a user to move easily between different simulators. To facilitate such tasks would require decoupling the interface from the internal representation, and provide an independent (neutral) data model as the basis for specification and storage of instantiated models; within such a standard framework for manufacturing simulators, translators could be added which would convert the model specification to the internal representation of target simulators. The overall approach is shown in Fig 4.41.

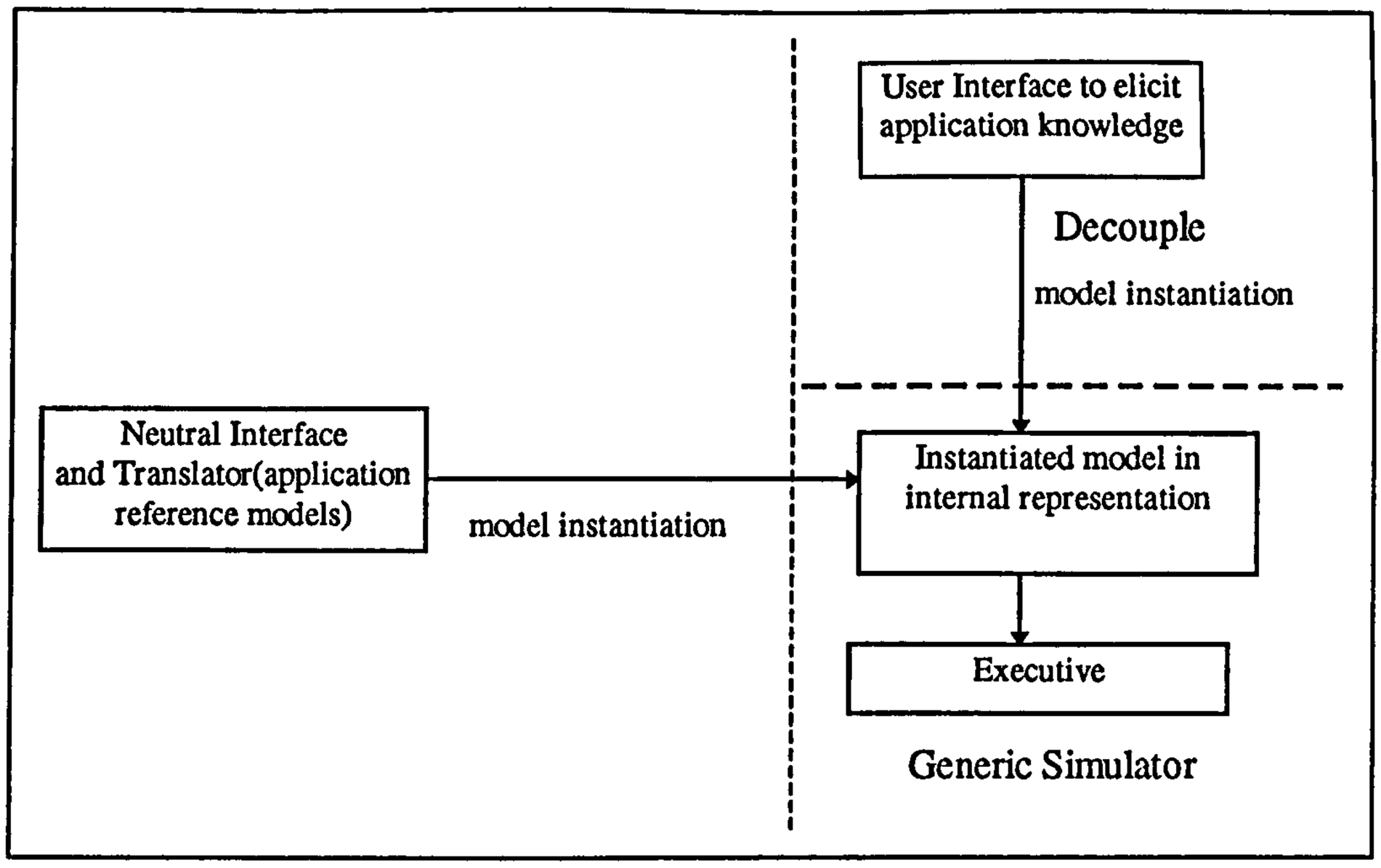


Fig 4.41 Framework for de-coupling data model from executive

The approach is broadly similar to that taken in STEP (Trapp, 1991; Burkett and Yang, 1992; Yang, 1993) in which a standard data model (modelling language and data representation scheme) was developed based on an analysis of the features and purpose of product design models. Translators, called application reference models in STEP, could then be added to convert the standardised data representations into models in target systems; a common approach for the application reference models was developed, but each one of them would need to be based on the protocols of the target system

In the following chapter, work done towards the implementation of such a standard framework is discussed. The framework is not a new simulator; it does not have a simulation executive of its own. Nor does it provide the facilities for specification and development of a full simulation model, but it is

designed to enable the representation of the common modelling elements of manufacturing simulators; the model specification would need to be further enhanced in the specification language of the target simulators. Maintenance of a minimum model specification through the standard framework would reduce substantially the effort needed in rewriting it for another simulator, if the need arises; it, however, provides ample opportunity for vendors of such systems to differentiate their products in the full range of modelling capabilities provided, in the ways the simulation executive works and how reports are presented.

Chapter 5 Standard Manufacturing Simulator Framework

5.1 Overview

In this chapter we provide a description of the development of a Standard Manufacturing Simulator Framework (SMSF), based on the analysis of requirements for such a framework and the common elements of generic simulators presented in the previous chapter . A schematic diagram of the framework is shown in Fig 5.1, and it comprises three distinct elements:

1. an interface for the user to specify the application model; it can be viewed as providing the user with a tool for mapping the common data requirements of a simulation model onto its internal representation in SMSF;
2. an internal data representation, called the Common Representation of Manufacturing Simulators (CRMS); and
3. a module containing translators for converting the CRMS data into the corresponding representation of target manufacturing simulators; this is currently illustrated with translators to generate (partial) models in three popular simulators and, in addition, a natural language description of the application model.

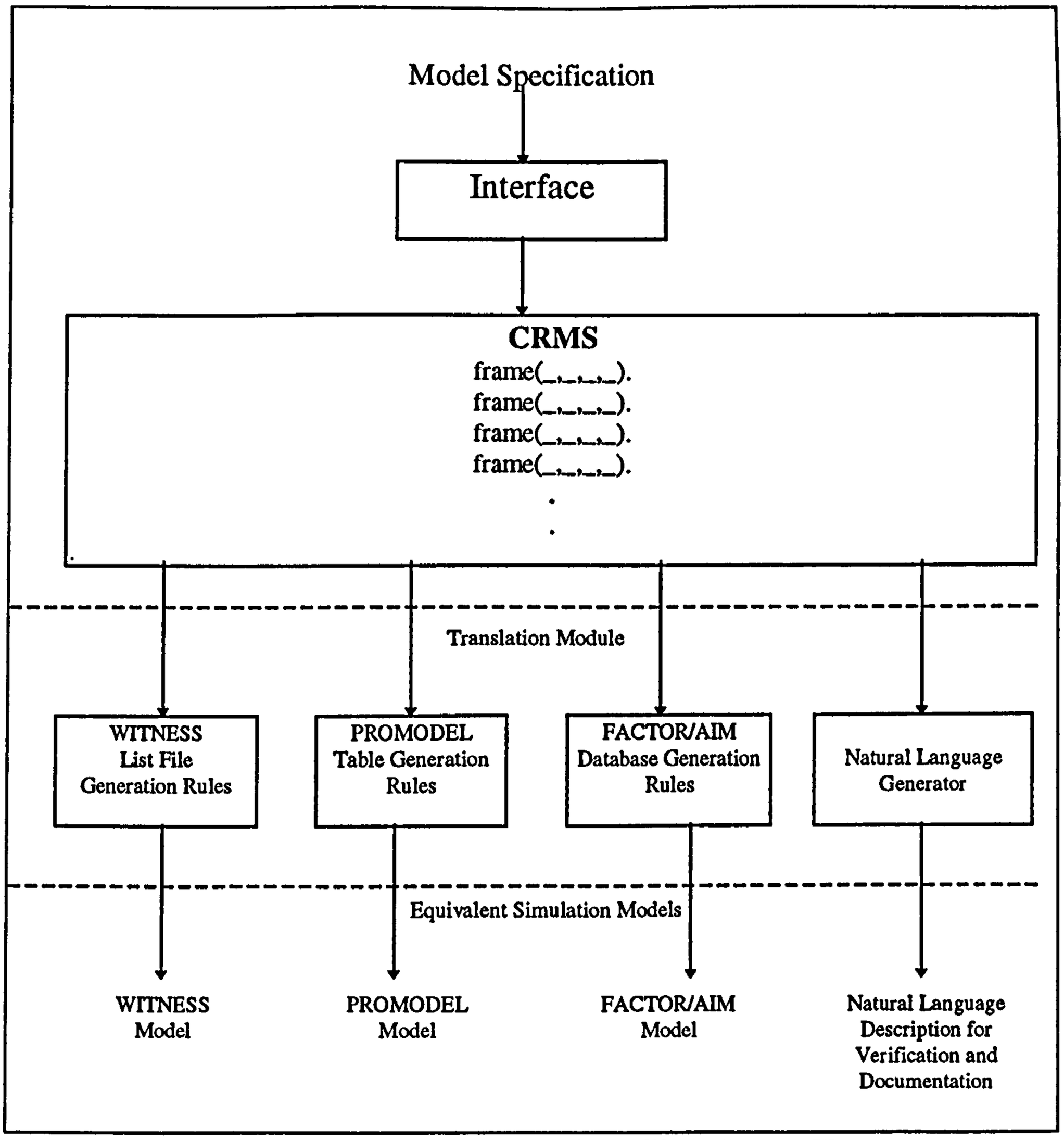


Fig 5.1 The Standard Manufacturing Simulator Framework

CRMS is at the core of the framework and devised specifically as an internal data representation for the common modelling elements identified in the previous chapter, and is independent of any generic simulator. It is in the form of a frame structure, that has been used to represent information about the definition and details of entities.

CRMS is instantiated for an application via a dialogue interface which provides the user with a number of dialogue boxes. Each dialogue can contain a single question, a number of questions or combination of questions and list box selections; depending on the selections made within a dialogue, other dialogues are activated. Incremental data entry is possible so that if a model specification needs to be added to or modified, the user and the internal analysis process do not have to start from the beginning.

Partial model generation in target simulators is performed by translators which use the instantiated CRMS as data. These translators are production rules which fire when their left hand sides match the CRMS data; when the production rules fire, their right hand sides are activated and perform actions which write the simulation models. Its operation is illustrated with translators (application reference models) that generate for three simulators: WITNESS, PROMODEL and FACTOR/AIM. The model generated format for: WITNESS is a list file; PROMODEL is a text file in the form of a number of tables; FACTOR/AIM is a database comprising tables for different entities.

An English language generation facility which translates the CRMS data into a natural language specification of the model is also included. This is useful for verification and documentation purposes so that the user has a record of what he has specified, and ensure that he has entered what he intended; the latter is especially useful when there is a large model to be entered requiring vast amounts of information via a long sequence of dialogues.

The software for the system has been written in Prolog. The main reason for the selection of this language was that it is widely used in the area of AI and automatic simulation programming. The actual implementation, LPA Prolog, proved useful in that it allowed dialogues to be implemented using a portable

dialogue manager. The Prolog stream system is particularly useful because it allows the writing of program statements to a file using the BIP (Built In Predicate) **write**. The Prolog language is entirely backward chaining meaning that it is goal directed, which is adequate for the specification process whose goal is the elicitation of a complete specification and the instantiation of CRMS. The framework, however, also required the implementation of a forward chaining inference mechanism for model generation, which is a data-driven process; it uses the model representation in CRMS as data and the WITNESS, PROMODEL and FACTOR/AIM model generation rules as production rules. This was achieved using Oops, a forward chaining inference mechanism implemented in Prolog.

5.2 User Interface

The user interface elicits the model specification for a given manufacturing system, and instantiates the internal data representation in SFMS. CRMS and the user interface together provides the data model of SFMS that is used by the application specific translators (or reference models) to generate partial models in any generic manufacturing simulator. As discussed in the previous chapter it contains knowledge of manufacturing systems but excludes any knowledge of simulation; the latter remains within the target simulators.

5.2.1 Specification Methods

A number of methods exist for specifying models, in general, and for manufacturing systems in particular. They each have both advantages and disadvantages, and have been used in various automatic programming

systems. A brief discussion of the principal contenders help to present the reasoning behind the particular approach that was selected for use in SFMS.

Natural Language Interfaces

The use of a natural language interface involves specifying the model using the constructs (syntax, vocabulary, grammar) of a natural language in the form of a number of sentences. As an example the time between arrivals of a part could be specified as:

The time between arrivals of part 2 is according to an integer uniform distribution with a minimum value of 10 minutes and a maximum value of 15 minutes.

The advantages of the approach are:

1. it is the closest representation to the human way of thinking.
2. provides vocabulary and syntax.
3. little training is needed for learning input requirements
4. provides easy to read self documentation

The disadvantages are:

1. constrained input, making them inflexible; attempts at increased flexibility result in reduction of ease of use and excessive training requirements.
2. for modelling large systems, the specification will often exceed the size of the generated program.

Dialogue Interfaces

Dialogue interfaces employ (Murray system, 1986) dialogues containing one or a number of fields for input to elicit data. Due to development of more advanced interface technology, it is usually possible for all the characteristics of a specific element of a manufacturing system to be entered within a single dialogue. As an example, a dialogue for part characteristics could have fields for: arrival time of first part; time between successive arrivals; lot size; maximum number of arrivals.

Advantages of the approach are:

1. least labour intensive, and user friendly since the user is prompted and guided through the specification.
2. little or no training required.

Disadvantages of the approach:

1. if the specification progresses through a large number of dialogues, the user may find it difficult to remember what has been specified.
2. inability to backtrack to earlier dialogues for modification or correction of input.

Graphical Interfaces

Graphical interfaces are used to specify the layout of a system and are used in conjunction with a natural language or dialogue interface. They can be special purpose (Sinclair, Doshi, and Madala, 1985; Raczynshi, 1990; Gong and McGinnis, 1990) and restricted to a specific domain, or general purpose (Koshevis and Chen, 1980) applicable to any simulation domain.

Advantages of the approach are:

1. Most suitable for systems for which pictorial representation is desired or necessary

Disadvantages of the approach are:

1. The special purpose graphical interfaces are useless outside very restricted domains and often require knowledge of directed graph theory.
2. The general purpose graphical interfaces, although they may be applicable to any simulation domain, require more expertise for model specification.

5.2.2 Choice of approach

Dialogues were chosen as the specification method because of their ease of use. Also, recent developments in windows based dialogues allow a number of characteristics to be entered in a single dialogue, for example, the majority of the characteristics of a specific part; this is advantageous from the point of view of both system development and speed of specification. The development

of these user friendly dialogues has become easier with improvements in computing technology (e.g. GUI, etc).

Natural language was deemed unsuitable because it can only be used in a restrictive manner as a formal high level language, which would require learning; also it is more labour intensive to use with little system guidance which is inherent in dialogue interfaces.

Graphical interfaces are usually more difficult to develop, and often restrictive in their application; since the requirement here is for data entry rather than pictorial layout, graphical interfaces were not used.

5.2.3 The Dialogues

In this section a brief illustration is provided on the use of dialogues to elicit from a user the specification of an application model. This is presented with examples for entering data on the main components of a manufacturing system, i.e. parts, transportation and machines. The sequence in which the dialogues appear is shown in Fig 5.2; particular emphasis is placed on the part and machine dialogues, with the transportation dialogues shown within a single box for conciseness, and only dialogues for the specification of a part are discussed in detail.

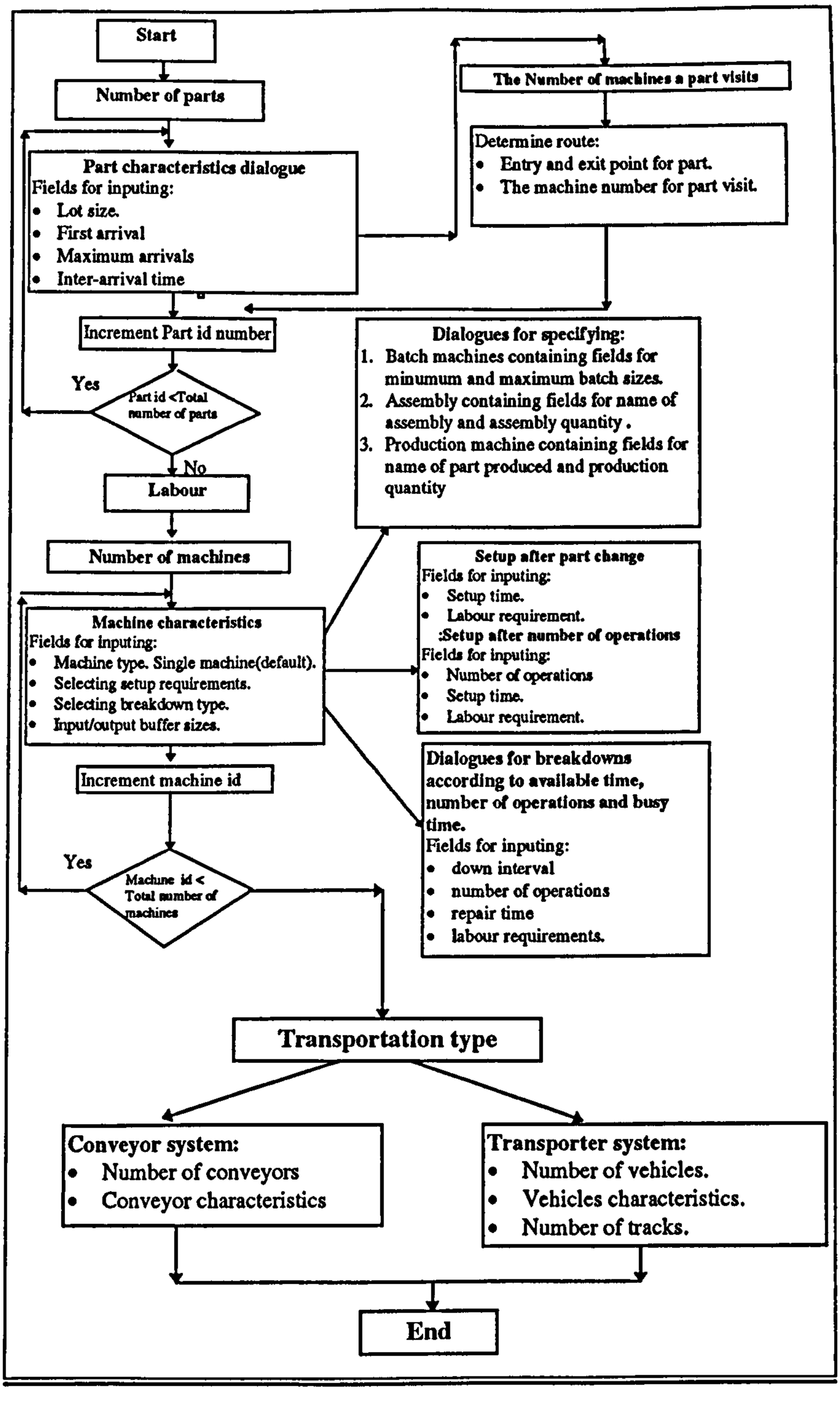


Fig 5.2 The sequence of dialogues for elicitation of the model specification

Part specification dialogues comprise three different sections.

The first dialogue shown in Fig 5.3 determines the number of parts in the system.

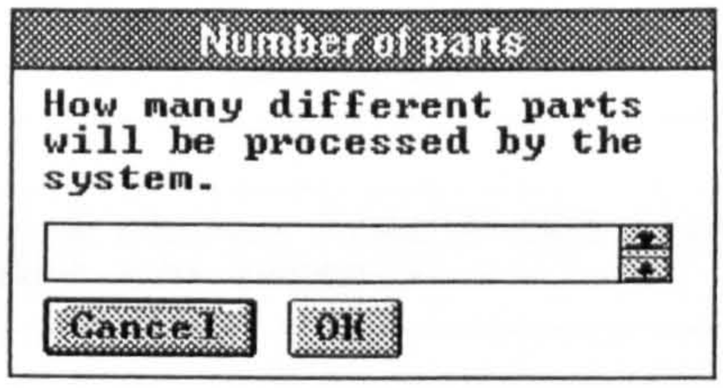


Fig 5.3 Dialogue to elicit number of parts

The second dialogue shown in Fig 5.4 elicits the characteristics of the individual parts.

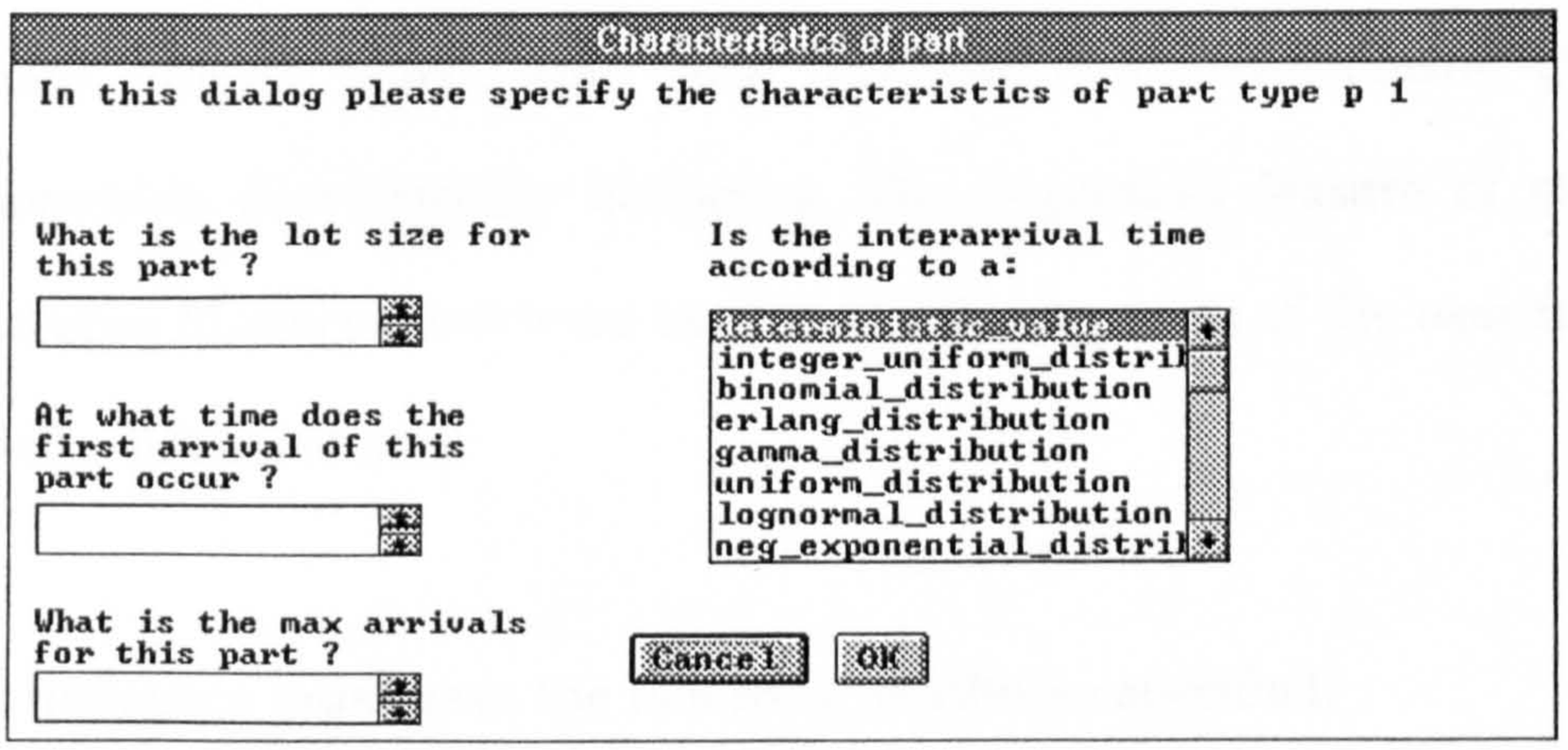


Fig 5.4 Dialogue to elicit part characteristics

Depending on the selection made in the inter-arrival time list box in the previous dialogue, a dialogue to elicit parameters of the selected probability

distribution appears. For example a dialogue to establish the parameters of an Erlang distribution is shown in Fig 5.5.

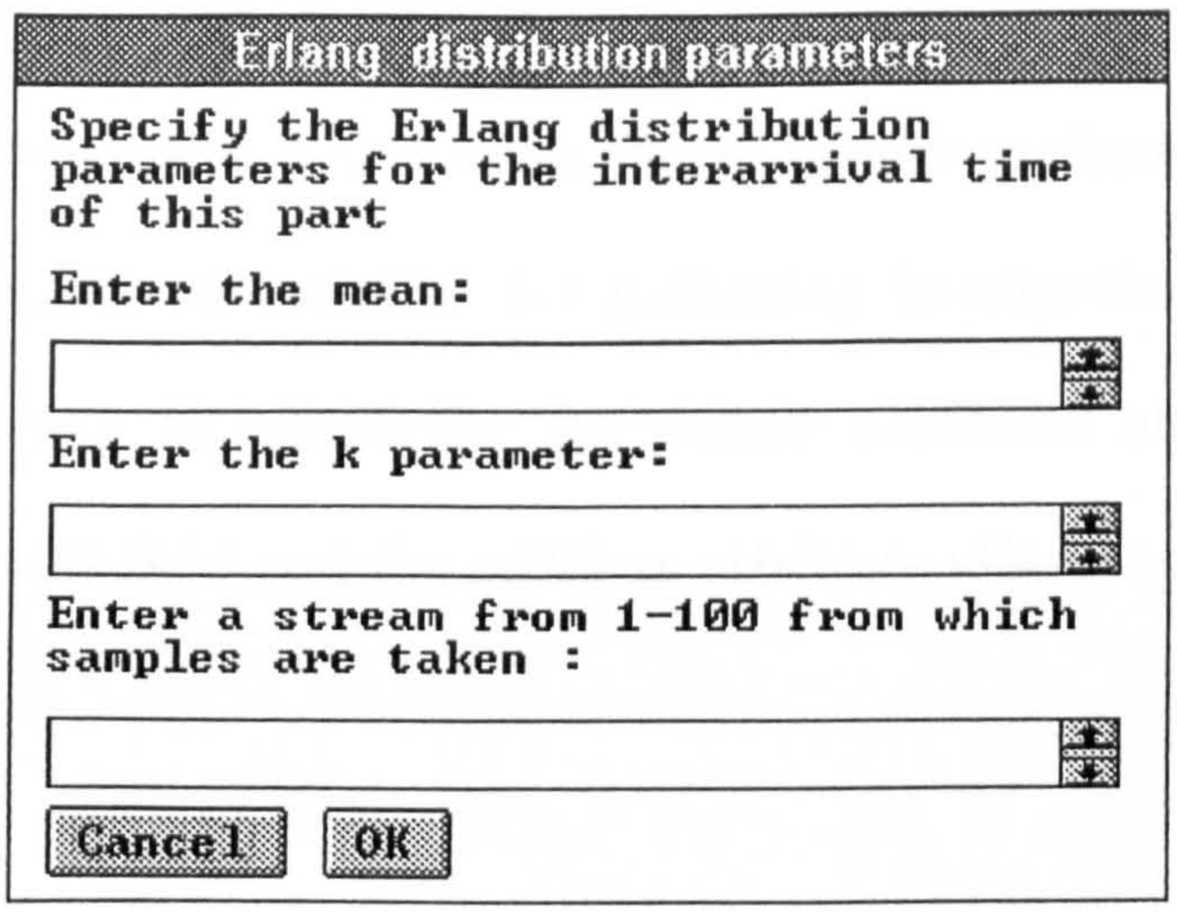


Fig 5.5 Dialogue to elicit parameters of Erlang distribution

The dialogues were developed using the Portable Dialogue Manager in LPA Prolog, which allows the rapid and efficient development of Windows compatible user friendly dialogues. The important feature of the Portable Dialogue Manager that were used in the development of the user interface are briefly discussed here.

All dialogues must have the following attributes specified:

1. **Invoke.** This attribute specifies that the dialogue is used to gather information from the user, and will have termination buttons like OK, CANCEL or ABORT.
2. **Title.** The title attribute of a dialogue definition is used for representing the title characters of the window in which the dialogue is drawn.

Each dialogue can have a number of different field types specified.

There are four types of fields which are:

1. **text fields** which constitute a region for presenting information to the user or, when specified as editable, for gathering information from the user. These fields may be scrollable. Its **value** attribute allows data to be included in a text field and the ***editbox*** attribute allows a user to type data into a dialogue.

In addition attributes can be used which are common to the Four Fields. These are:

- The **name attribute** of a field is specified as an argument of the field type, and is used to identify a particular field within the dialogue.
- the **location attribute** specifies the position of a field absolutely or relative to some preceding field.
- The **size attribute** is used to represent the width and height of field.
- The **return attribute** of a field return is used to return the list of item(s) selected by the user when presented with options in the dialogue.
- A field can be positioned below another field using the **below attribute.**

2. **picture fields** which constitute a non-functional region into which a pre-defined bitmap image of a picture can be drawn.

3. **button fields** to represent the control aspects of a dialogue. The *cancel* and *ok* buttons are attributes of this field.
4. **list fields** comprising a list of scrollable Prolog atoms. There are two attributes specific to this field:
   - The **select attribute** which is used to state whether a single or multiple choice selection is expected from the user in response to the dialogue.
   - The **layout attribute** is used to specify the layout of the dialogue, e.g. layout (scrollbar) for a box with a scrollbar attached, or layout (popup) for a popup menu, or layout (Rows*Colums) for a grid of control buttons.

As an example the predicate for generating the characteristics of a part is ***diapart_3***:

```
diapart(A,B,C,D,E)::=
invoke(modal),                                  %sets up dialogue as modal
title('Characteristics of part');               %specifies title of dialogue
text(t1),size(512,50),                          % specifies t1 as text field
value('In this dialogue please specify the characteristics of part type p')% writes text
                                                    in t1

text(t9),across,size(50,10),                    %specifies t9 as text field across from t1
value(A);                                       %writes part number after 'p' of text field t1
text(t2),style(label),size(200,30),below(t1),   %specifies t2 as text field below t1
value('What is the lot size for this part ?');  %writes text in t2
editbox(t3), size(150,20), below(t2,0),return(B); %specifies t3 as editbox below t2
                                                    instantiating B with value
                                                    entered
text(t4),size(200,40),below(t3,20),             %specifies t4 as text box below t3
value('At what time does the first arrival of this part occur ?');  % writes text in t4
editbox(t5), size(150,20), below(t4,0),return(C); % specifies t5 as editbox below t4
                                                    instantiating C with value
                                                    entered
text(t6),size(200,25),below(t5,20),             %specifies t6 as text field below t5

value('What is themax arrivals for this part ?');%writes text in t6
editbox(t7), size(150,20), below(t6,0),return(D); % specifies t7 as editbox below t6
```

```
                                        instantiating D with value
                                        entered
cancelbutton,at(240,220);               %specifies cancel button and location
okbutton,across;                        %specifies ok button across from cancel
text(t8),size(200,30),right(t2,60),     %specifies t8 as text field right of t2
value('Is theinterarrival time according to a:');   %writes text in t8
list(dist),                             %specifies dist as list field
layout(scrollbar),                      %specifies that field has scroll bar
select(one),size(200,100),              %specifies that single selection list box
below(t8),                              % specifies that dist is below t8
return(E),                              %instantiates D with selection made
value(['deterministic_value',           %specifies list box choices
'integer_uniform_distribution','binomial_distribution',
'erlang_distribution','gamma_distribution','uniform_distribution',
'lognormal_distribution','neg_exponential_distribution',
'normal_distribution','poisson_distribution','random_value',
'tnormal_distribution','triangular_distribution','beta_distribution',
'weibull_distribution']).
```

5.2.4 Editing and Extending the specification modes

A particular disadvantage of the use of dialogue interfaces in the past has been the need to start a session from the beginning whenever a mistake was detected in the model specification, or if it needed to be modified/extended in any way; unlike natural language specifications, which allow data to be incrementally added or changed, dialogue interfaces used in the early simulation model generators were strictly sequential in nature. In SFMS, an editing and extending mode has been added to the user interface to allow the user greater flexibility in instantiating CRMS data, thus considerably adding to its ease of use.

The dialogue shown in Fig 5.6 is used to indicate to the system whether the user wants to edit the model or enlarge it.

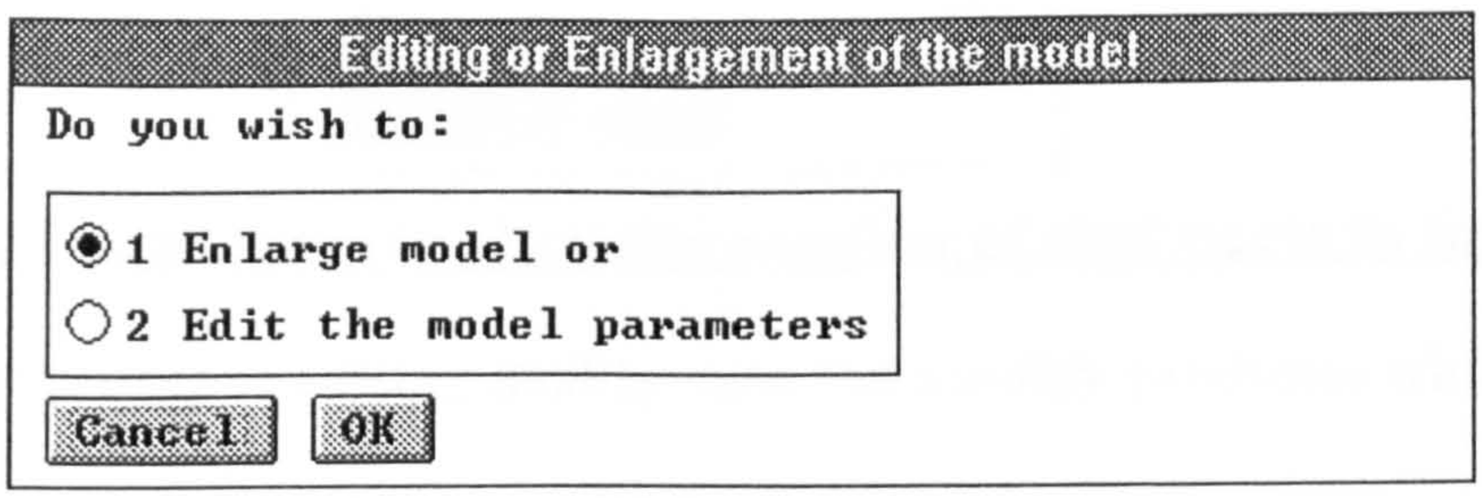


Fig 5.6 Dialogue for model modification

If the enlarge model button is selected then the Model Enlargement dialogue, shown in Fig 5.7, is used to elicit which components are to be added to the existing model.

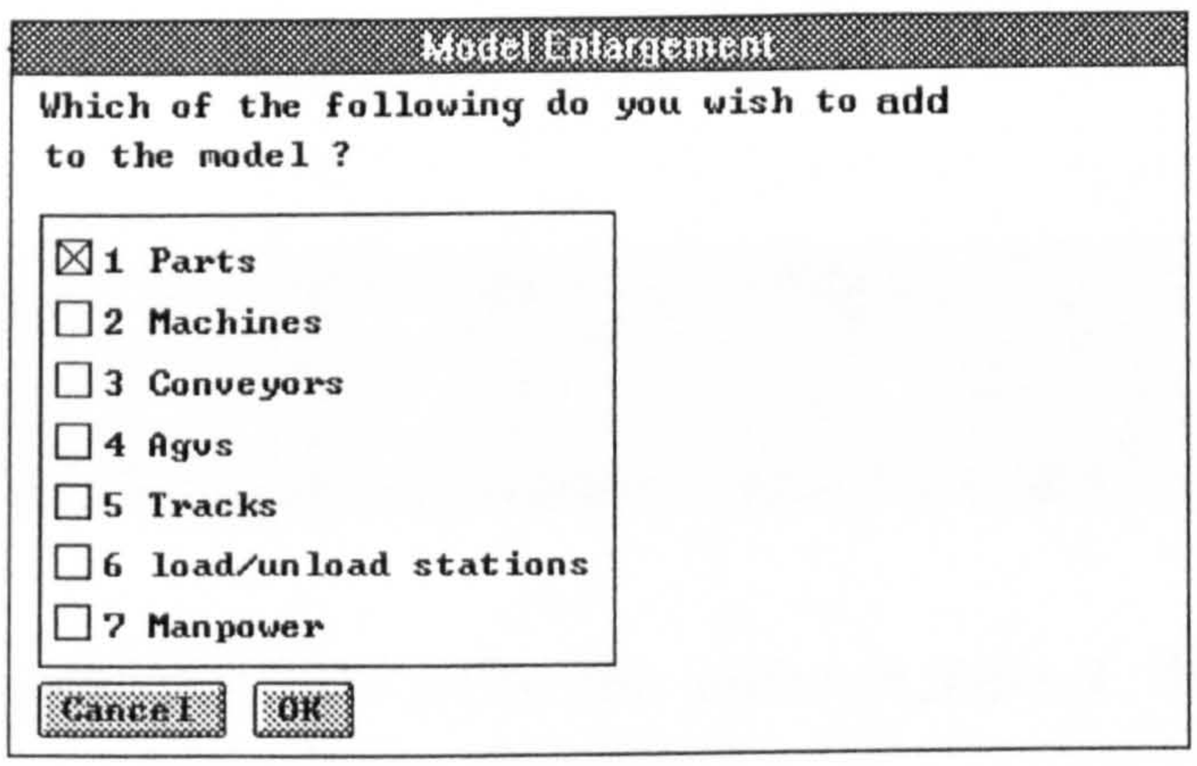


Fig 5.7 Dialogue for model enlargement

If ***Parts*** box is selected with a cross hair in the ***Model Enlargement*** dialogue then the dialogue to elicit the number of parts, shown in Fig 8, is used to obtain the number of new parts which are to be added to the CRMS. Once the number of additional parts is determined then the part characteristics dialogues, previously explained, is used to elicit their individual characteristics.

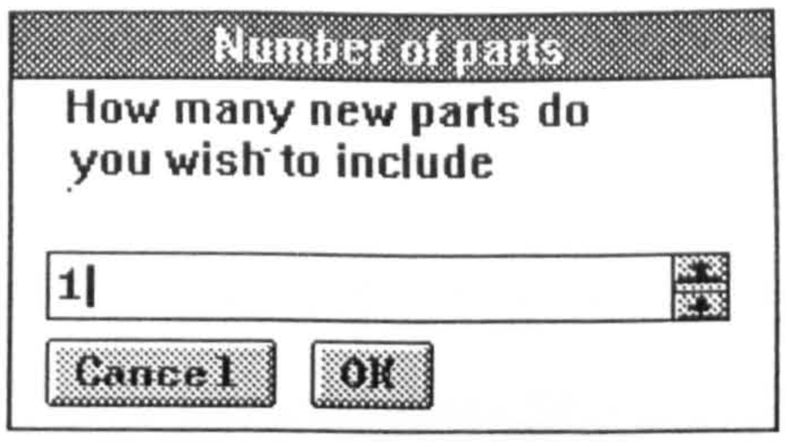


Fig 5.8 Dialogue to elicit the number of new parts to include

This enlargement or editing facility uses the ***modify*** predicate which calls the ***edit_or_enlarge_1*** dialogue predicate

```
modify:-edit_or_enlarge(X),
wclose(edit_or_enlarge_1),
test_modify(X).
```

where ***test_modify***, which comprises two clauses, is used to determine whether the user wants to enlarge (call predicate enlarge) or edit (call predicate edit) the model.

```
test_modify(X):-X='1 Enlarge model or'->enlarge.

test_modify(X):-X='2 Edit the model parameters'->edit
```

The ***enlarge/0*** predicate calls the ***enlarge_type_1*** dialogue to determine which component(s) is to be added to the model. It then uses ***test add_1*** to determine which components are selected for addition to the model.

```
enlarge:- enlarge_type(X),
wclose(enlarge_type_1),
test_add(X).
```

The ***edit/0*** predicate calls the ***edit_type_1*** dialogue to determine which component(s) are to have their characteristics modified. It then uses ***test_edit*** to determine which individual component's characteristics need to be edited.

```
edit:- edit_type(X),
wclose(edit_type_1),
test_edit(X).
```

If the part information needs to be edited, the ***test_edit_1*** clause below succeeds, and is used to elicit the part number and access its existing characteristics from the clause ***store***. These existing characteristics are removed from the clause stores using the ***retractall*** predicate, and the new characteristics elicited and added to CRMS using the ***diapart_5*** (dialogue for elicting part characteristics) and ***add_4*** (adds characteristics entered in dialogue to CRMS) predicates. Similarly ***test_edit*** predicate is used to modify characteristics of machine, conveyor, AGV, manpower, etc.

```
test_edit(A):-
member('1 Part information',A)-> % tests to see if part characteristics
                                   have to be modified
which_part(B),% If answer is yes then dialogue 'which_part_1' is called to
                determine which part. Its part number is instantiated with B.
wclose(which_part_1),          %closes dialogue 'which_part_1'
load_files('M:\PARTINFO.PL',          %loads part characteristics section of CRMS
        [if(true),load_type(source),all_dynamic(true)]),
retractall(fact(frame(creation_info,B,Y,val,Z))), % deletes old characteristics of part B
del('M:\PARTINFO.PL'),
tell('M:\PARTINFO.PL'),
listing(fact/1),
diapart(B,E,F,G,H),% calls 'diapart_5' to elicit new characteristics of part B
wclose(diapart_5),
add(creation_info,B,lot_size,val,E),      % Adds new characteristics of part B
add(creation_info,B,first_arr_tim,val,F),  to CRMS
add(creation_info,B,max_arr,val,G),
add(creation_info,B,inter_arr_tim,val,H),
elicit_and_ass_int_arr_dist_par(B,H),
fclose('M:\PARTINFO.PL'),
del(Head,A,Rem),
test_edit(Rem).
```

The ***test_add_1*** predicate used for adding characteristics of new parts operates, in a similar manner to the ***test_edit_1*** predicate.

5.3 Internal data representation

In the previous chapter we examined the nature of generic simulators, and three in depth, to identify a number of common modelling elements with characteristics. In this section work on the development of a representation for these common elements in a generic simulator independent format, called the Common Representation for Manufacturing Simulators (CRMS), is discussed.

5.3.1 Knowledge Representation

A number of AI knowledge representation methods exist and were considered for the development of CRMS. Knowledge representation can be defined as a method of encoding in a communicative manner knowledge about the real world or its status via languages, descriptions or pictorial representations. For effective knowledge representation, a stylised version of the real world must be encoded via the use of a formal language. These formal languages are commonly referred to as knowledge representation schemes.

5.3.1.1 Types of knowledge

A distinction needs to be made between factual and inferential knowledge in relation to the purpose of any study which, in this case, is manufacturing simulation modelling. This enables a distinction to be made between the simulation model representation of the manufacturing system and the expertise required to create the simulation model.

Factual Knowledge

Factual knowledge treats a collection of knowledge as a static model involving the extraction and storage of large amounts of data that show some property, and is closely associated primarily with information science in which the emphasis is on using computers for classifying, indexing and searching through data. It can be thought of as declarative knowledge comprising a static collection of facts and a set of general procedures for manipulating them. This knowledge is compatible with the representation of a simulation model, which is a static representation of data from the real world. A factual representation is declarative because it is viewed strictly as data with no consideration of how the data will be used.

Inferential Knowledge

Inferential knowledge is concerned with the reasoning behind the creation of the factual data, instead of simply its classification and retrieval. It can be thought of as procedural knowledge since it involves the execution of a number of steps to reach conclusions from a number of basic facts. Inferential knowledge is an addition to declarative knowledge, in the form of a program to manipulate the declarative knowledge. In most AI environments this inferential knowledge can be a number of logical assertions, which can be used in addition to declarative knowledge to form a complete program. A theorem can then be used to execute the program for problem solving. Logical assertions can be represented in a number of programming languages, with PROLOG (Roussel, 1975; Clocksin and Mellish, 1984; Bratko, 1986) perhaps the most universally accepted.

Inferential knowledge can be seen from the simulation perspective as the reasoning and knowledge required to create a simulation model, i.e. model development and encoding. An inference mechanism is a necessity for applying inferential knowledge so as to create factual knowledge. In the case of simulation modelling this would involve examining the real world in relation to simulation modelling expertise, via human reasoning, to develop a simulation model.

5.3.1.2 Knowledge Representation Methods

Predicate Logic

This is a formal language to allow conceptualisation of the real world. For example we can represent the statement that a **part 1 is concentric** in predicate logic as:

concentric(part_1)

where concentric is the predicate and part_1 is the constant. It can also be used to represent relationships between objects. For example the statement a **machine 1 is an assembly machine** can be represented as:

isa(machine_1,assembly_machine).

It is also possible to use standard logic symbols in predicate logic formulae like:

→(implication),¬(not),∨(or),∧(and),∀(for all),∃(there exists). This allows representation of statements like:

1. All parts require some pallet as ∀x:∃y:requires(x,y).

2. machine 1 is not lathe as ¬not(machine_1,lathe)

Examples of rules would be a statements such as:

1. All gears are concentric as ∀x:gears(x) →concentric(x)
2. A mass production system is a manufacturing system which has high production volumes:
   ∀x:manufacturing_system(X)∧high_volume(X)→mass_production_system (X).

In Prolog this would be written as:

mass_production_system(X):-manufacturing_system(X),high_volume(X).

Predicate logic is a universal abstract language for representing knowledge, whereas logic programming is a sub-set of predicate logic. Logic programming allows situations to be described with the formulae of predicate logic, and with the aid of a problem solver can be used to make inferences from the formulae. Prolog is the best known logic programming language.

The search strategy used in Prolog is termed backward chaining and is an example of goal directed problem solving; for example, to prove the goal that a manufacturing system is mass production, we must first prove the sub-goals that it is a manufacturing system and it is high volume. The alternative to this search strategy is forward chaining or data-driven inference. These react to

changing data and are sometimes called production systems; for example the rule: ***If part one has been processed,*** call AGV could be represented as:

condition	**action**
If part processed	**→call AGV**

Semantic Nets

This representation attempts to describe concepts or objects and was first used as a software representation by Quillian (1968) in the area of natural language processing. This is achieved by analysing world meanings and the way they interact with one another. Semantic networks comprise a net of nodes, which represent concepts and meanings, and links which represent relationships between nodes. Examples of ***nodes*** can be **MACHINE, PART, CONVEYOR, ASSEMBLY_MACHINE**, and ***links*** can be **IS, IS_A, DO, HAS, CAUSES.** Semantic network of an ***assembly machine*** is shown in Fig 5.9.

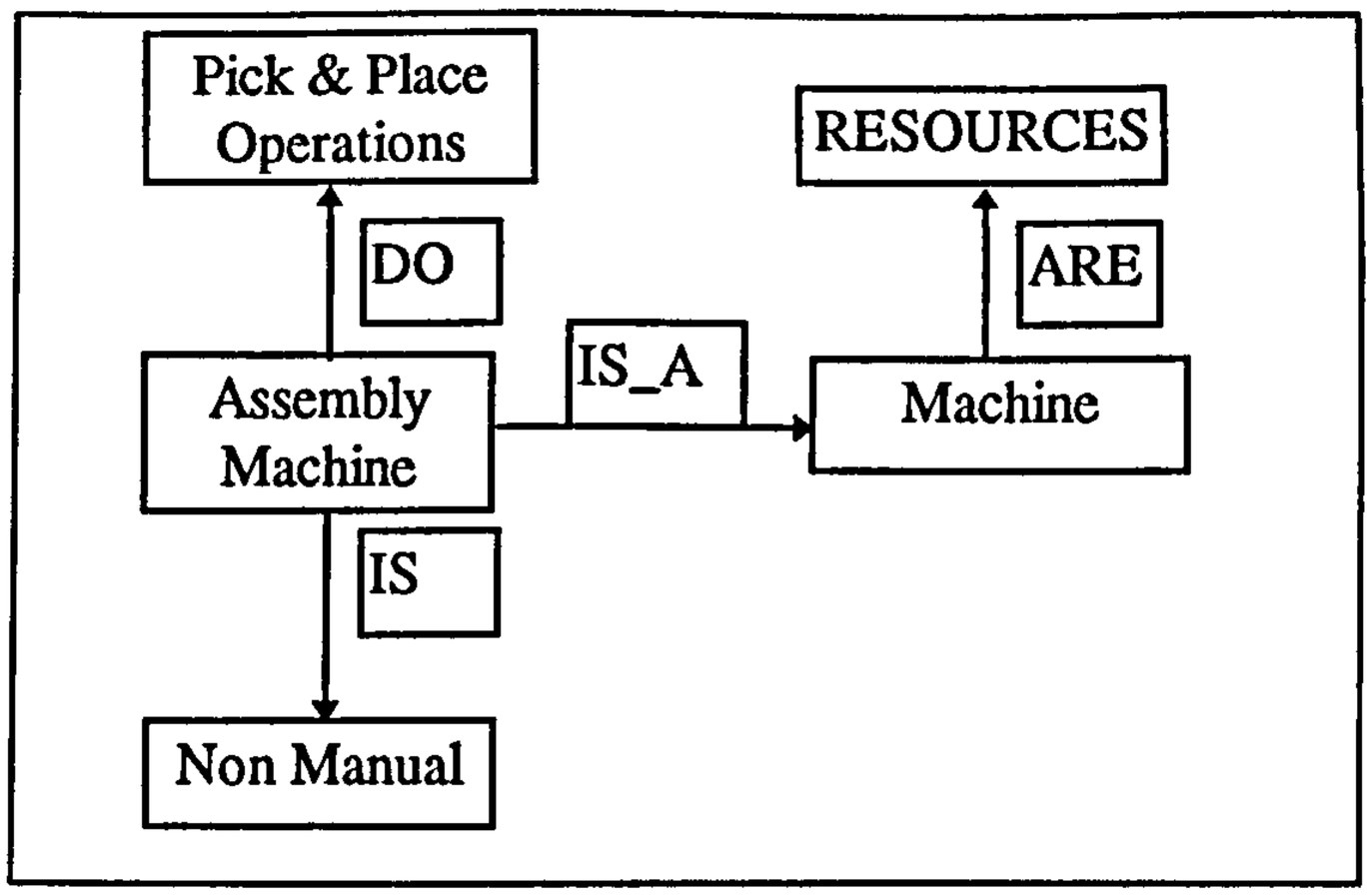


Fig 5.9 Semantic Network to represent an assembly machine

This can be represented in symbolic logic as:

IS_A(assembly_machine, machine)

IS(assembly_machine,non_manual)

DO(assembly_machine,pick_and_place_operations)

ARE(machine,resources)

However, as mentioned by Ringland and Duce (1988), the use of nodes and links can have certain disadvantages if the designers of networks are careless in the way they assign meanings to nodes. For example a type node labeled "machine" cannot make a distinction between the class of all machine or a typical machine.

The Frame Knowledge Representation Scheme

The development of frames as a knowledge representation scheme alleviated the problem of distinguishing between classes and their specifics by providing a mechanism for representing stereotypical concepts or events. This is achieved by allowing properties to be set at the higher levels of a hierarchy which can then be inherited by those at lower levels. They were developed as a means for representing knowledge about objects and, as a concept, were initially envisaged by Minsky (Minsky, 1979). They differ from rule based systems in that they are geared to representing in a structured way a large of number of facts. They can be thought of as a data structure comprising a number of components called slots, which have unique names and contain specific types of information like:

- A specific value.
- A pointer to other frames.
- A procedure for computing the slot value.

An example of a frame for storing knowledge of ***conveyors*** is shown below:

frame:conveyor
a_kind_of:transportation
moves_by:mechanical_means
activity_during:working_shifts

where the slots are a **a_kind_of**, **moves_by**, and **activity during**. These have slot values **transportation**, **mechanical_means** and **working_shifts**

respectively. This could be represented as the prolog clauses containing four arguments: as

```
frame(conveyor,a_kind_of,value,transportation)
frame(conveyor,moves_by,value,mechanical_means)
frame(bird,activity_during,value,working_shifts)
```

where the:

- 1st argument is the name of the frame,
- 2nd argument the slot name,
- 3rd argument indicates the type of the data the slot holds and is a specific value, and
- 4th argument holds the slot entry which is a specific value in this case but could be a pointer to another frame or a procedure.

Scripts

Scripts (Schank and Abelson, 1977) are a specialised structure often used to represent sequences of events in a restricted domain, whereas frames are more general purpose structures for the representation of common clusters of facts.

For example scripts can be used to define a typical situation at a machine cell (Schank and Abelson, 1977) such as when a ***part enters*** the cell, when its ***operation begins and ends***, and when ***it exits***. This situation can be summarised by ***four*** main events as shown in Fig 5.10.

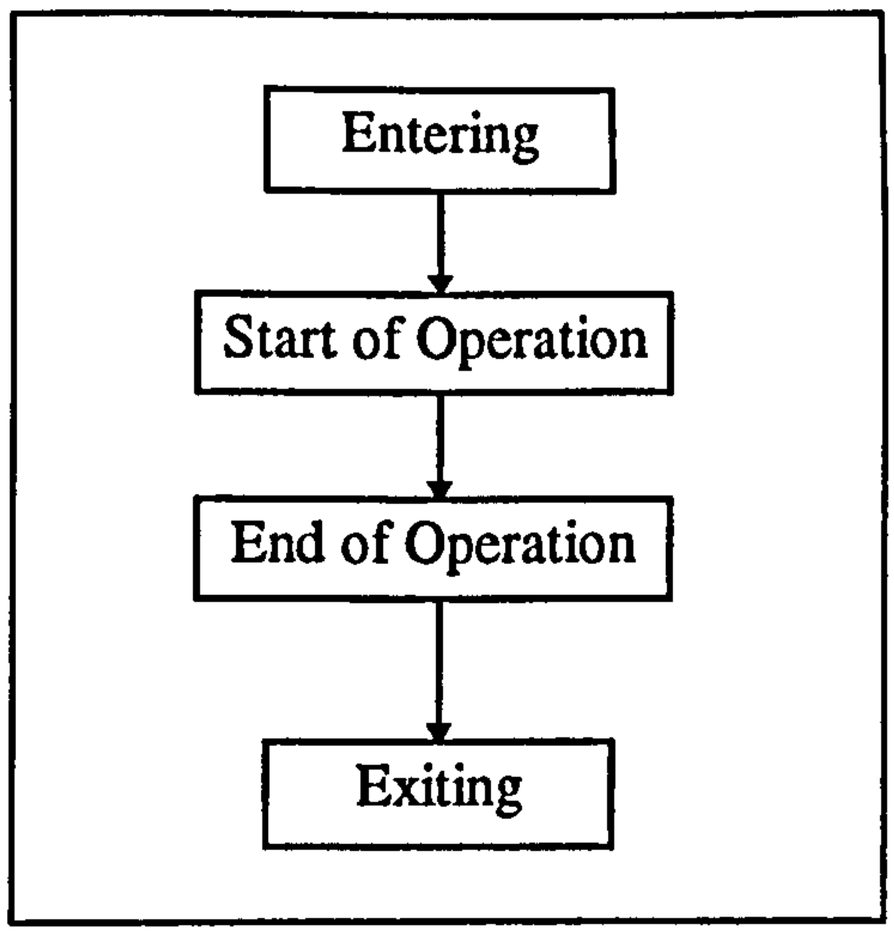


Fig 5.10 Events in machine cell script

Scripts have a number of important components such as:

- **Entry conditions**-Conditions that must be satisfied before events in a script can be executed. For example in the case of the machine cell, a part must have been scheduled before the script for it can be executed..
- **Results**-Conditions that prevail after an event in the script has occurred; for example, a record that the scheduled operation has been completed.
- **Props**-Objects like tools, fixtures, etc. which are involved in the events.
- **Roles**-Manpower which are involved in the events.
- **Track**-The actual sequence of events.

A frame could be used for representing the ***entering*** event as:

location:machine cell

action:enter machine_cell

locate an available machine

choose tool

locate on machine for processing

Object Oriented Programming

In object oriented methods (Tokoro and Ishikawa, 1984; Zaniolo, 1984), the basic computation entity is the object, and computation is only allowed via the passing of messages between objects. Each message has attached methods or procedures which perform computations to answer the message.

Objects can be arranged in classes to form a hierarchy of classes, where objects lower down in the hierarchy inherit characteristics and methods from those higher up. The only means of computation is to send a message to an object. There are two types of variables: class variables common to all objects in a class; and instance variables unique to a particular object.

A class hierarchy for ***automated materials handling*** is given in Fig 5.11. If a message was sent to the **AGV_1** asking about its type, it is handled by the method of the AGV class returning the answer ***automated***. If the same message was sent to the **conveyor_1** object, there is no class method in the conveyors class that can process the message, so it is sent to its superclass automated materials handling which uses its class method to return ***automated.***

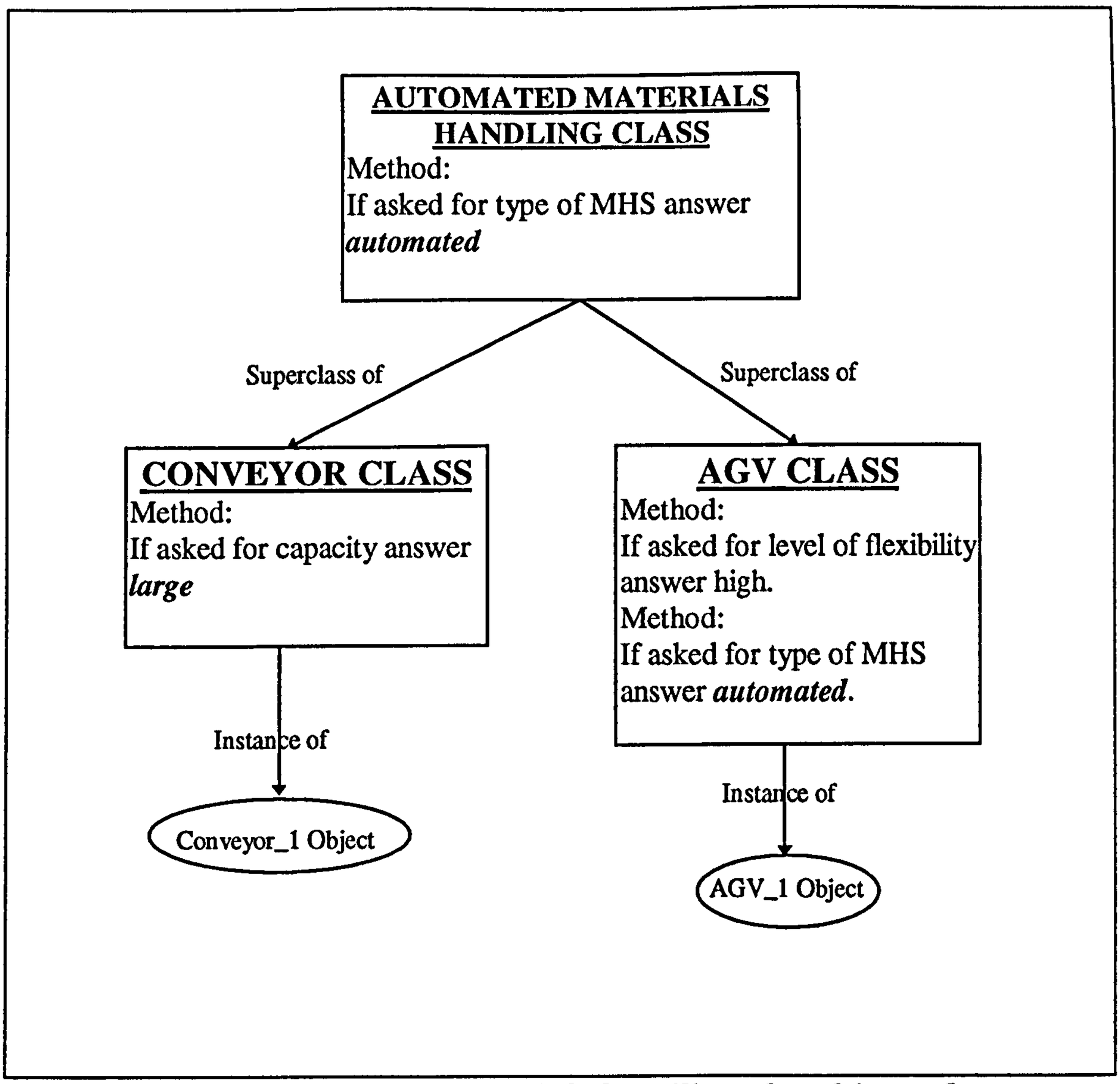


Fig 5.11 Automated materials handling class hierarchy

5.3.2 Knowledge Representation Scheme Selection

The previous discussion shows the wide ranging choice available for knowledge representation. Each of the methods, however, have both advantages and disadvantages and they have been developed for a different range of applications. In our framework, the objective is to choose a knowledge representation scheme for manufacturing system knowledge, which can then

be translated by an application reference model to the internal representation of any selected simulator. Simulation knowledge would be contained in the simulators and not in the framework.

Scripts were not considered suitable for use in the development of CRMS because not only manufacturing knowledge but also simulation knowledge would have had to be built into the framework; it would have involved the writing of a partial simulation model by translating the manufacturing system into the important script components like entry conditions, results, props, roles and track.

Since CRMS will be used to represent a declarative (factual) model of the real world, rules were discarded as a means of representation since they are used primarily for inferential knowledge e.g. simulation programming knowledge. The use of object oriented programming is unnecessary since our research deals with non-hierarchical modelling systems like WITNESS, PROMODEL and FACTOR/AIM, and not knowledge based simulation systems like ROSS.

Frames were considered the most suitable representation method since they are modular, with each type of entity (machine, conveyor, part, etc.) having a unique frame. For example, a processing resource entity (machine) could be represented by a frame called **machine**, containing different slots for representing the characteristics of different machines. In addition to their modularity, frame representations are easy to read, understand and modify.

Different frames can be used to represent different common modelling elements within which different slots can be used to represent their different characteristics.

5.3.3 The Frames of CRMS

In this section we illustrate how frames have been used within CRMS for representing the common modelling elements (data groups) and their characteristics, illustrated in Figs 5.12 to 5.19.

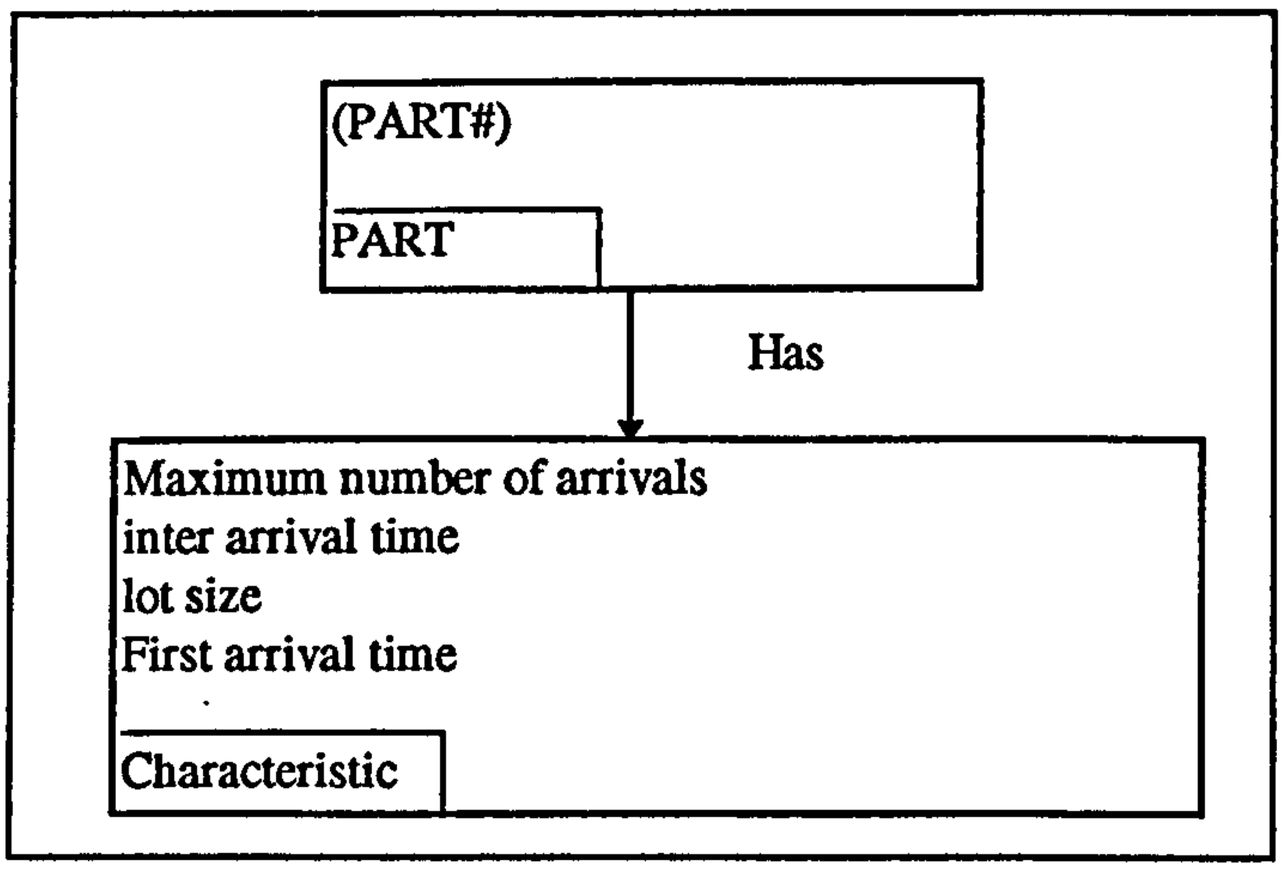


Fig 5.12 Part data group

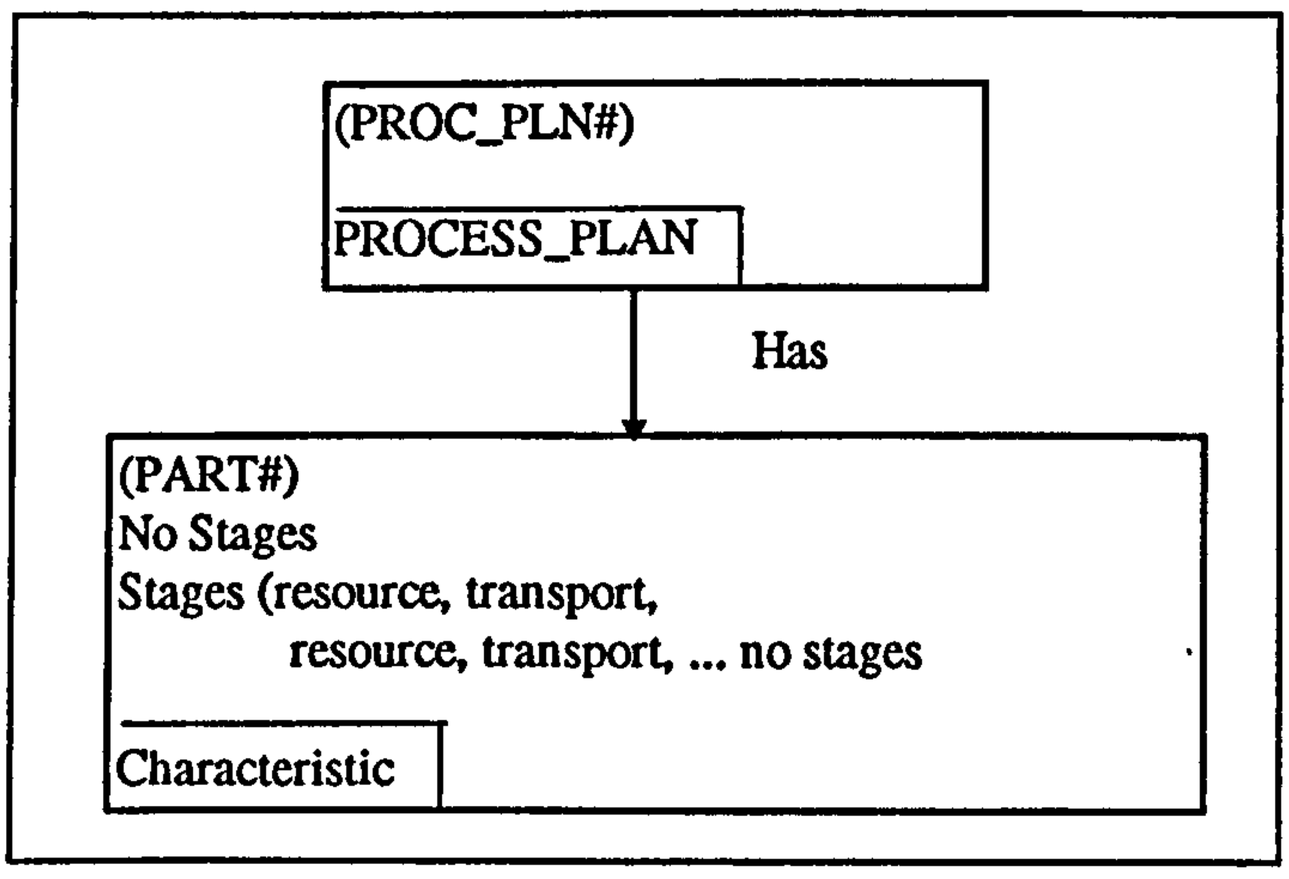


Fig 5.13 Process Plan data group

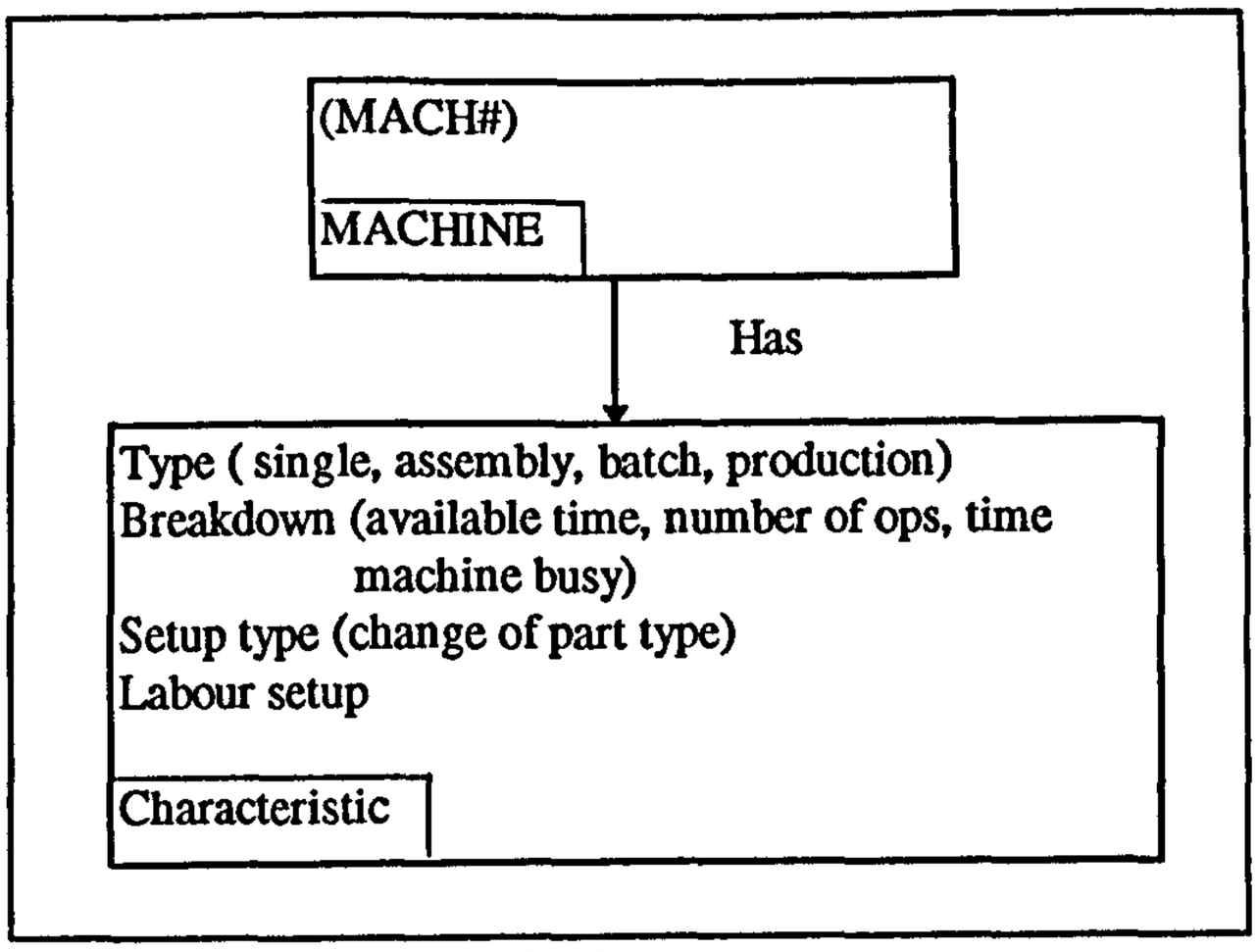


Fig 5.14 Machine data group

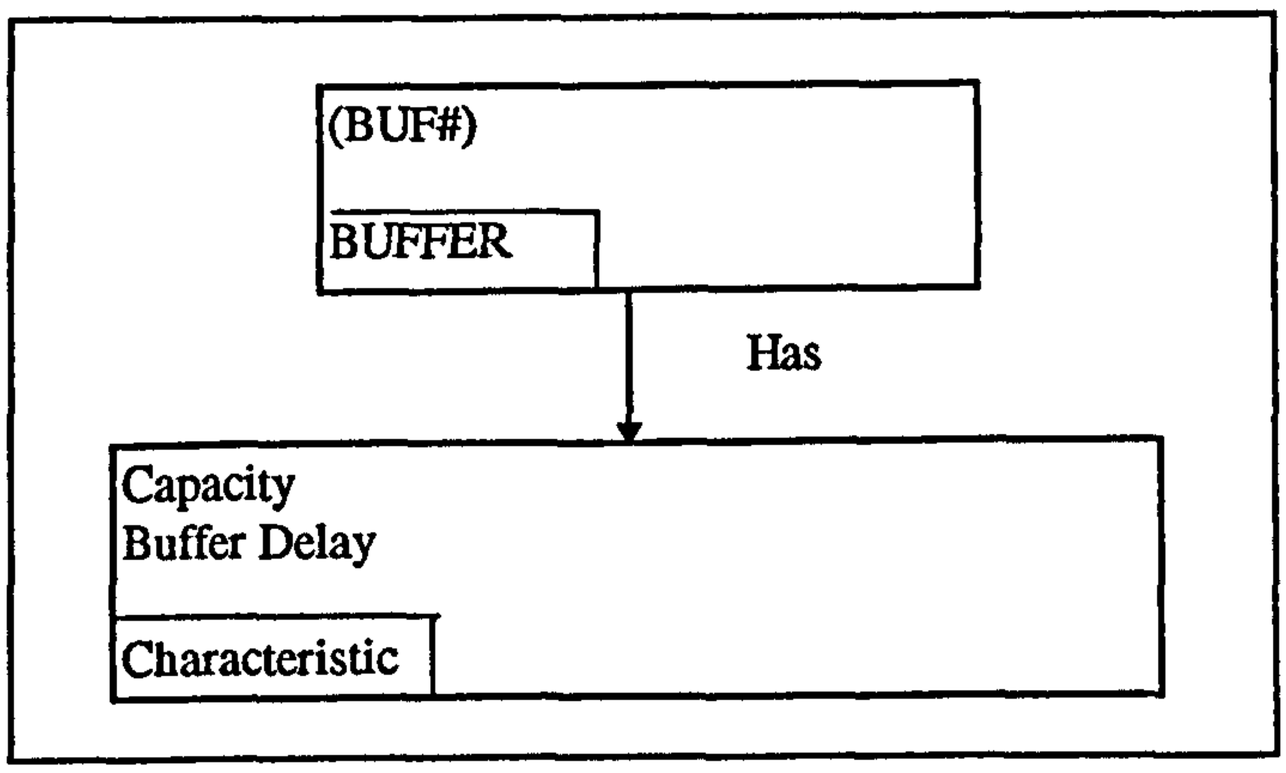


Fig 5.15 Buffer data group

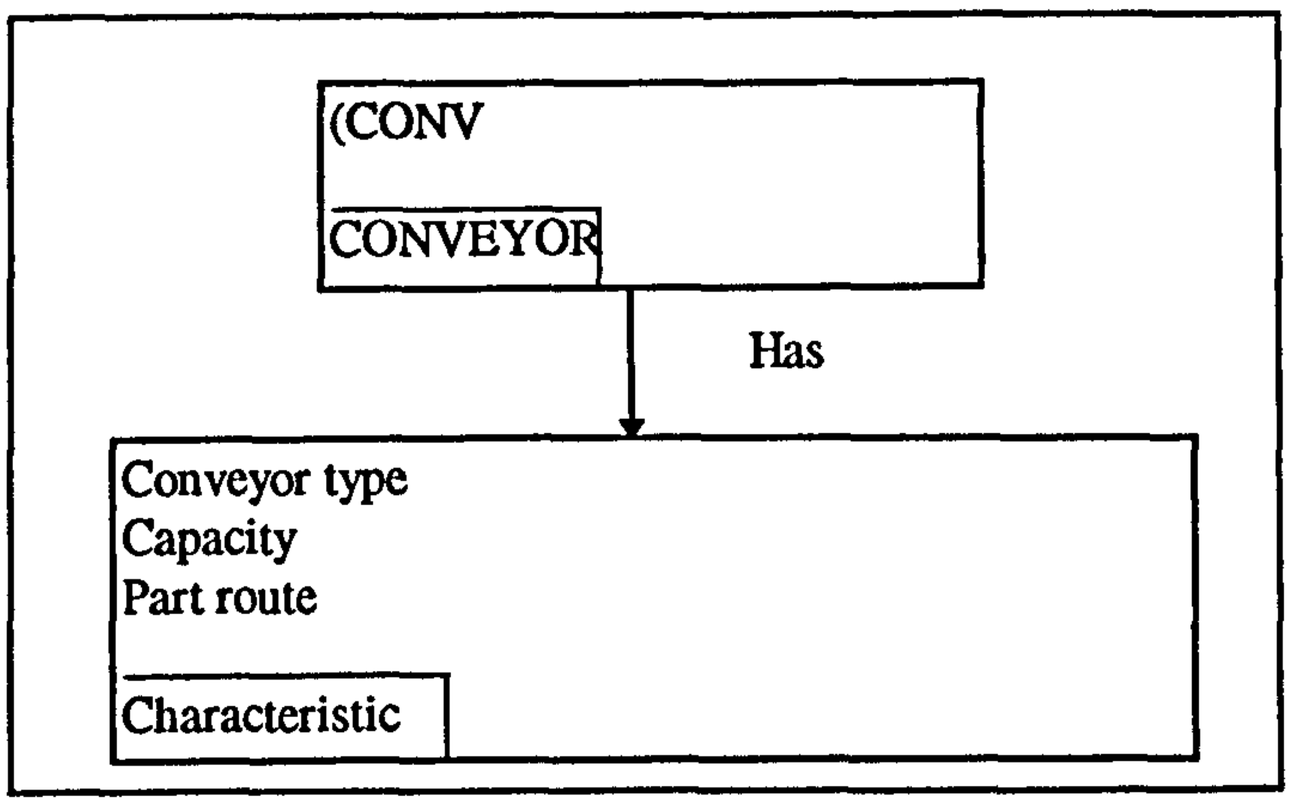


Fig 5.16 Conveyor data group

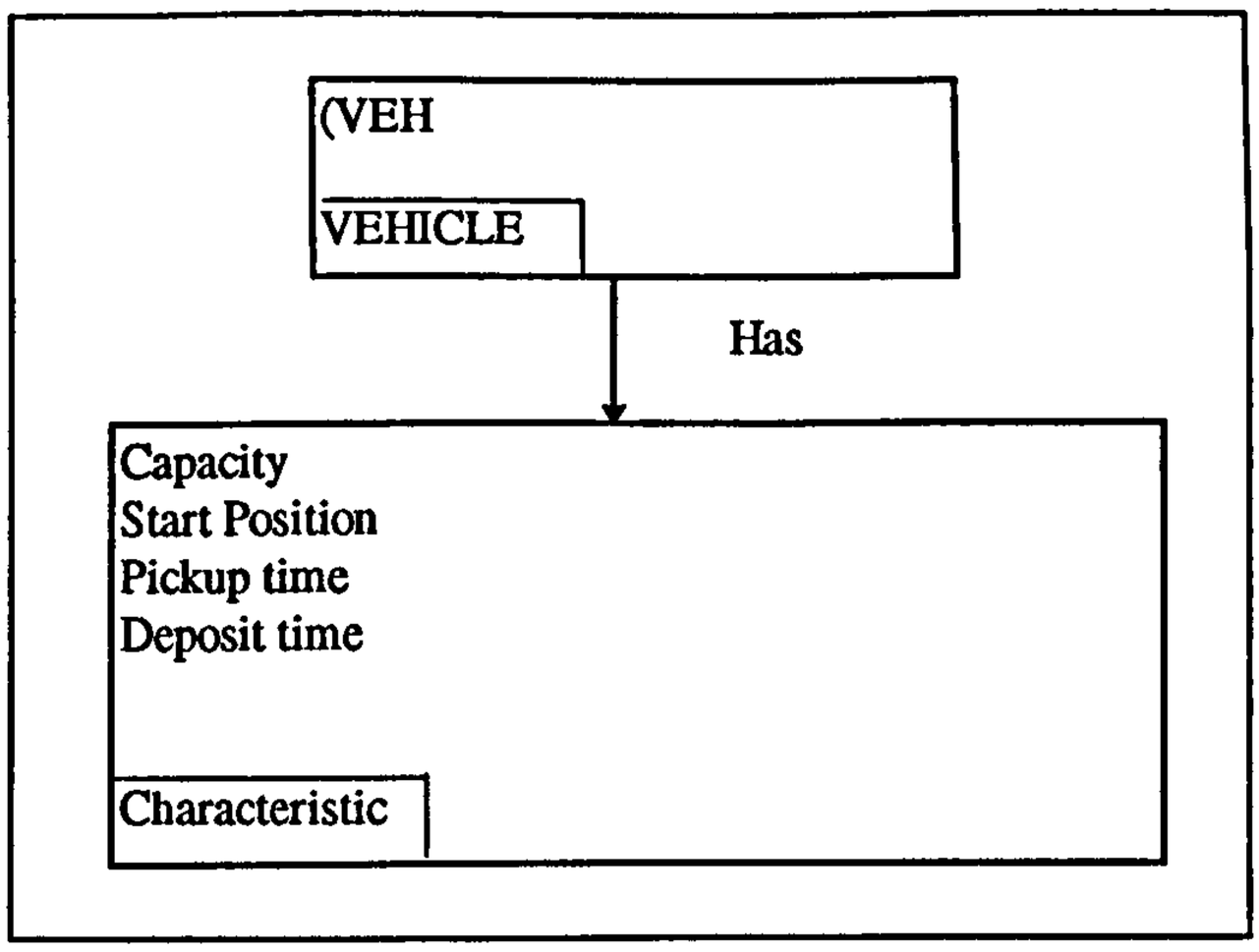


Fig 5.17 Vehicle data group

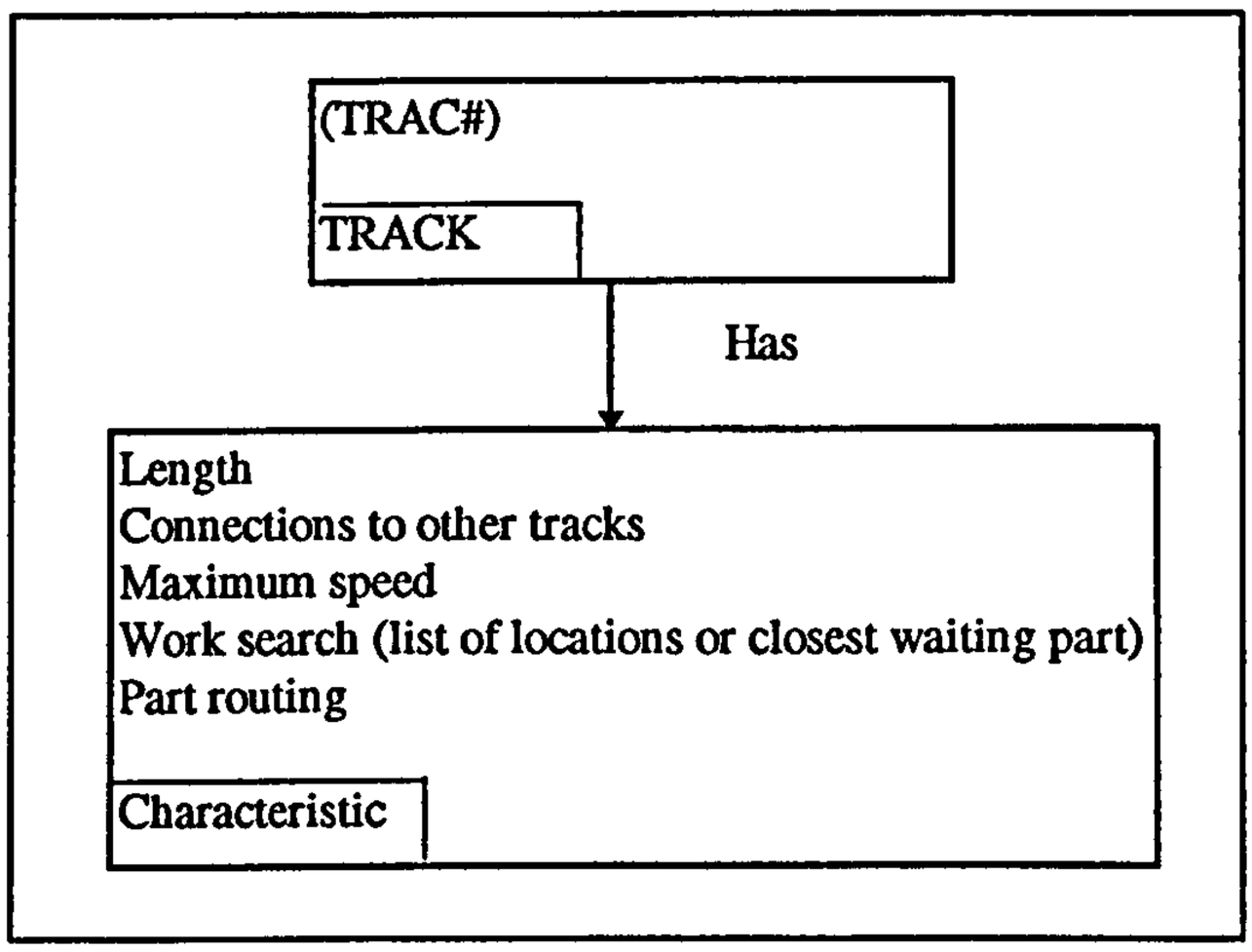


Fig 5.18 Track data group

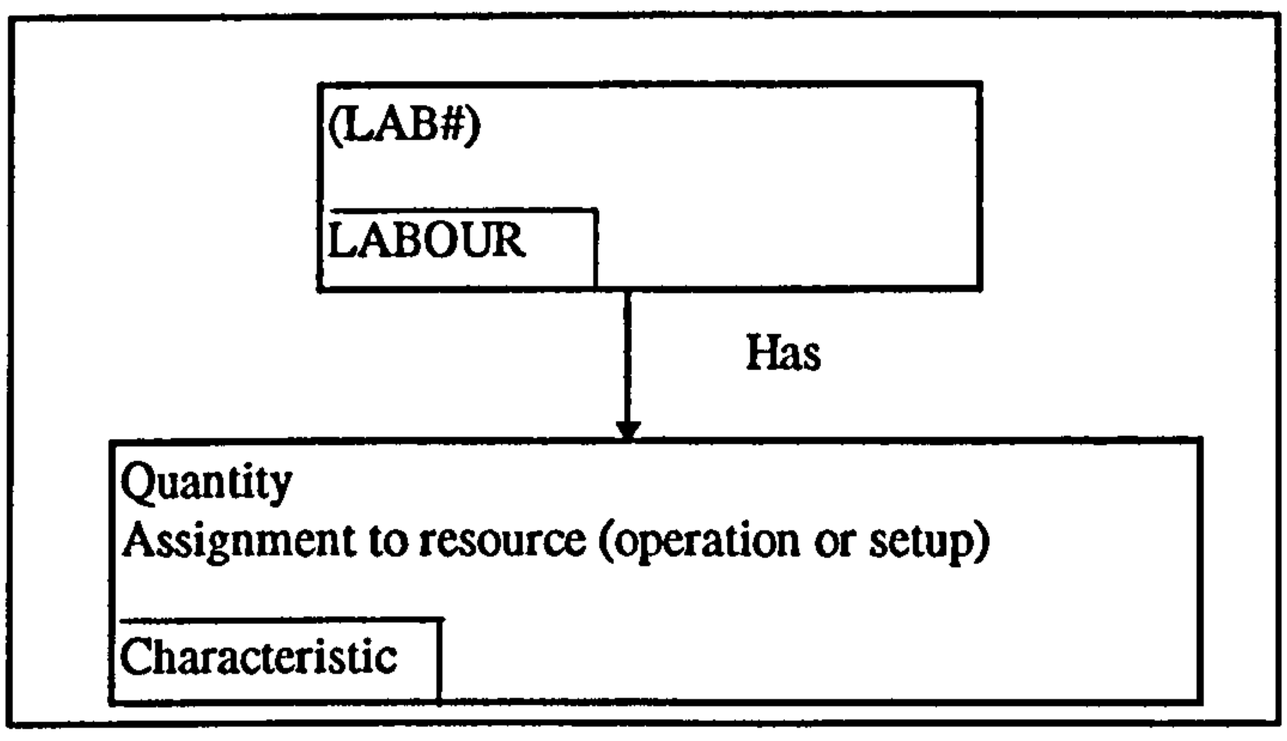


Fig 5.19 Labour data group

Only frames for parts, machines and part route are shown for illustrative purposes, but those for the other elements can be found in Appendix B. As discussed in the previous chapter only structural knowledge has been considered in the development of the framework; however, later we discuss how restricted types of behavioural knowledge may be added to it.

Parts

A part has a number of characteristics, as indicated in chapter 4, and in CRMS there is a slot associated with each of these. The general format of the frame used to represent parts in CRMS is:

frame(part,PART_No,SLOT_NAME, val, VALUE)

where:

- 1st argument is a constant denoted in Prolog by the atom 'part' and is the name.
- 2nd argument is a variable denoted in Prolog by the use of capitals and is a unique identifier for the part. For example part 1 would be denoted by PART_No=1.
- 3rd argument identifies a unique slot, corresponding to each of the sub-characteristics of parts, like: ***max_no_arr***(maximum number of arrivals), ***inter_arr_tim***(inter-arrival time), ***lot_size***(lot size), ***first_arr_tim***(time of first arrival). In addition there would be additional slots for representing probability distributions and their parameters.

- 4th argument is a constant denoted by 'val' to indicate that the slot contains a unique value as opposed to a pointer to another frame or a procedure for computing a value.
- 5th argument is a unique value of a slot. For example for a 'lot size' slot an entry 3 would indicate a lot size of 3.

For example in CRMS the inter-arrival time for part '1' according to a triangular distribution can be represented by the following frame entries::

```
frame(part,'1',inter_arr_tim,val,triangular)
frame(part,'1',p1,val,'2') )
frame(part,'1',p2,val,'4') )
frame(part,'1',p3,val,'6') )
```

SLOT_NAME=inter_arr_tim, p1, p2 and p3 in turn, where slots p1, p2 and p3 define the parameters of the triangular distribution

Machines

A machine has a number of characteristics, as indicated in chapter 4, and in CRMS there is a slot associated with each of these. The general format of the frame is:

```
frame(machine,MACHINE_No,SLOT_NAME, val, VALUE)
```

where:

- 1st argument is a constant denoted in Prolog by the atom 'machine' to identify the frame name,
- 2nd argument is a variable denoted in Prolog by the use of capitals and is unique number identifier for the machine. For example machine '1' would have MACHINE_No=1,
- 3rd argument identifies a unique slot such as: ***quantity*** (number of identical machines), ***type*** (machine type), the breakdown information, ***setup_type*** (type of setup) ***setup_time*** (time taken for setup), ***lab_setup*** (labour required to perform setup).
- 4th argument is a constant denoted by 'val' to indicate that the next argument requires a unique value as opposed to a pointer to another frame or a procedure for computing a value,
- 5th argument is a unique value of a slot.

For example, the number of identical machines is represented in the slot ***quantity*** as:

frame(machine,'1',quantity,val,'2')

where SLOT_NAME=quantity.

Part Route

Part route is separate from the ***part*** frame because it represents the interaction between temporary entities (parts) and permanent entities

(machine, conveyors, etc.). It provides the process interaction element of CRMS and, in itself, can be considered as one of the behavioural elements of a manufacturing system. The general format of the frame is:

frame(process_seq,PART_No,VISIT_No,SLOT_NAME, val, VALUE)

where:

- 1st argument is a constant denoted in Prolog by the atom '**process_seq**' to signify the frame name.
- 2nd argument is a variable denoted in Prolog by the use of capitals and is a unique identifier for the part. For example part '1' would result in PART_NO=1.
- 3rd argument is a variable denoting the visit number. For example VISIT_NO=4 to represent the 4th machine visited by a part.
- 4th argument is a unique slot used to identify: ***load_unload*** (load/unload station), ***machine_no*** (machine identifier), ***conveyor_no*** (conveyor identifier), ***transporter_no*** (transporter indentifier), ***transport_time*** (travel time), ***proc_time*** (processing time), etc.
- 5th argument is a constant denoted by 'val' to indicate that the next argument requires a unique value as opposed to a pointer to another frame or a procedure for computing a value.
- 6th argument is a unique value to represent a load/unload station, location identification, processing time or set up time.

For example if part 1 uses conveyor 1 to visit machine and is processed for 3 minutes it would have the following CRMS entries

frame(process_seq,'1',1,machine_no,val,'1')
frame(process_seq,'1',1,conveyor_no,val,'1')
frame(process_seq,'1',1,proc_time,val,'3')
frame(process_seq,'1',1,proc_time_units,val,'minutes')

where VISIT_NO=1 and SLOT_NAME=machine_no, conveyor_no, proc_time, process_time_units.

5.4 Model Translators

The third element in SFMS is the translation module for transforming data of an application model instantiated in CRMS into an internal representation of any target generic manufacturing simulator. The module, in practice, consists of indiviual translators (application reference models) for each generic system, and SFMS currently contains one each for developing representations in WITNESS, FACTOR/AIM and PROMODEL. The system could be extended to cover other generic manufacturing simulators since CRMS is designed to conform to the common elements and their characteristics of all such simulators. Rules are used to represent inferential knowledge in the translators because they attempt to create from factual knowledge in CRMS more factual knowledge in the form of a (partial)simulation model representation. Rules are often referred to as production rules since they produce new knowledge from existing knowledge. It is effectively knowledge required to create a simulation model, and is highly dependent on the language in which the model is to be generated, i.e. the rules for generating models in PROMODEL, WITNESS and FACTOR/AIM will be different and

unique dependent on the underlying internal data representation model of each.

In order to apply this knowledge, a forward chaining inference mechanism is required to apply the rules to existing knowledge in order to create new knowledge. These forward chaining inference mechanisms are commonly called production systems because when the left hand side of a rule matches working memory (CRMS) the actions (model writing) specified on the right hand side are performed. A forwarding chaining production system called Oops (Merritt, 1989) was implemented in Prolog as part of SFMS. The rule selection algorithm is simple; where the first rule matching working memory is selected.

Knowledge is encoded in Oops by creating rules with the following syntax:

rule <rule id>:

[<N>:<condition>,....]

==>

[<action>,.....].

where:

- **id**:- unique identifier for the rule.
- **N**:- optional identification for the condition-pattern to match against working memory. It should be noted that each of these conditions has to be a legal Prolog data structure, including variables.

The major predicates of Oops are shown in Fig 5.20.

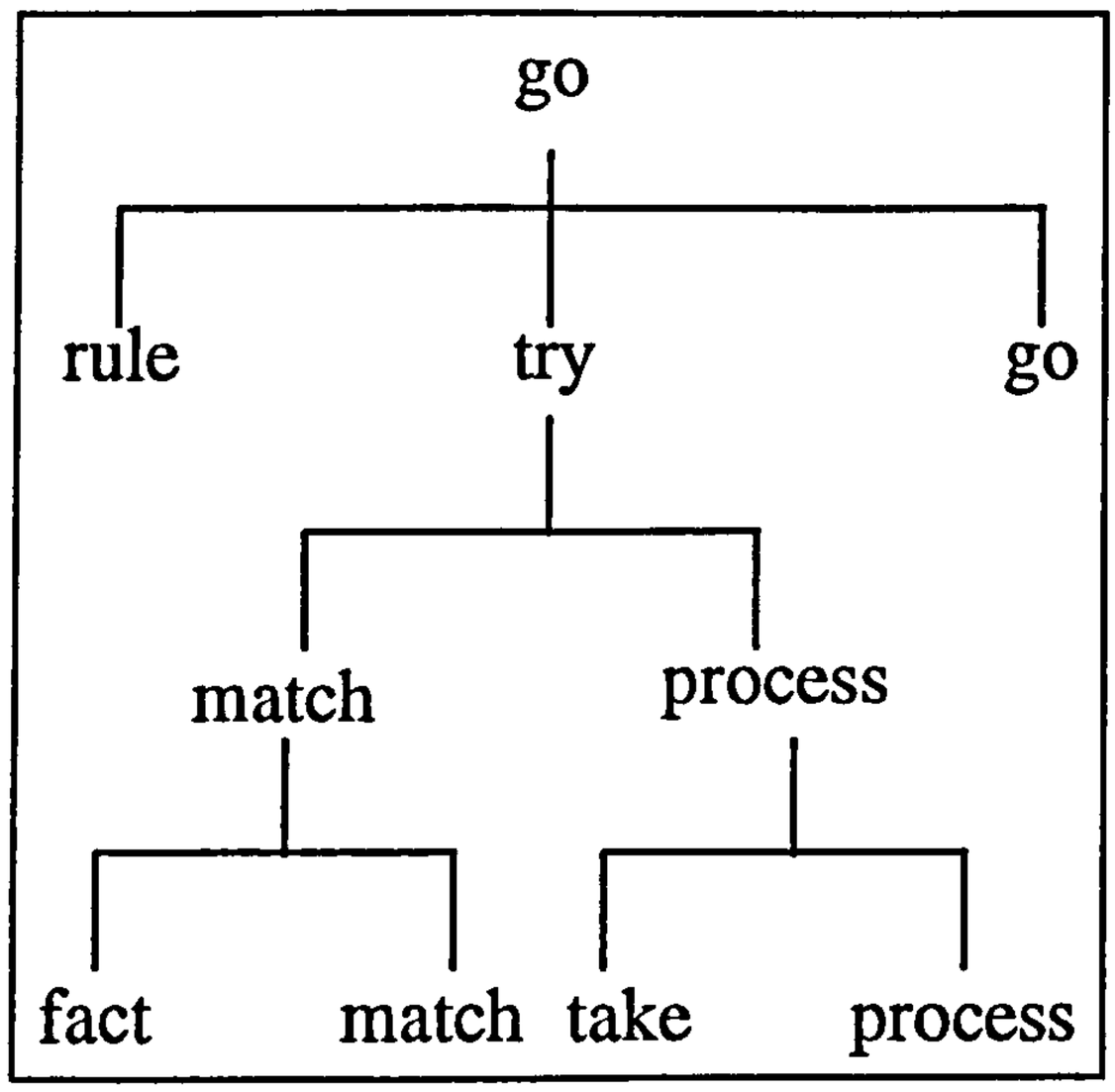


Fig 5.20 The major predicates in the Oops inference engine

5.4.1 Model Translation

In this section we illustrate how instantiated CRMS is transformed into equivalent WITNESS, PROMODEL and FACTOR/AIM internal model representations.

5.4.1.1 Translator for WITNESS

Structure of WITNESS

In WITNESS the internal model representation is in the form of a list file, which is a text representation of the model and is composed of two parts:

1. The ***define section*** which contains definitions of all entities in the model such as parts, machines, buffers, tracks, vehicles, conveyors, etc. A machine for example, is defined using a statement of the form:

```
MACHINE: m1,1,Single,0,0,3;
```

which means that machine m1 is of type ***single*** and of quantity 1. A conveyor definition statement such as:

```
CONVEYOR: convey1,1,Fixed,18;
```

means the system contains a conveyor called convey1 which is ***fixed*** and can hold a maximum of 18 parts.

2. The ***detail section*** which contains statements comprising the characteristics of the entities. For example the statement for detailing machine m1 may be:

```
m1

 NAME OF MACHINE: m3;
 QUANTITY: 1;
 TYPE: Assembly;
 * Assembly quantity: 3;
 PRIORITY: Undefined;
 LABOR:
   Repair: None;
 END
 LABOR:
   Cycle: None;
 END
 DISCRETE LINKS :
       Fill: None
 END
 DISCRETE LINKS :
       Empty: None
 END
```

```
SETUP_DETAIL
  Setup number: 1
  * Mode: Part change;
  * Setup time: R_SETUP;
  * Description: Setup number 1
  * Station number: 1;
  LABOR:
    Setup: None;
  END
END SETUP_DETAIL
CYCLE TIME: R_CYCLE;
BREAKDOWNS: Operations;
* Ops between breakdown: 250.0;
* Repair time: 66.0;
* Scrap part: Yes;
* Setup on repair: No;
INPUT RULE: BUFFER  (10);
OUTPUT RULE: IF ISTATE (m3) = 5
             PUSH to SCRAP
            ELSE
             PUSH to ROUTE
            ENDIF;
REPORTING: Individual;
SHIFT: Undefined,0,0;

END m3
```

Translation Methodology for WITNESS

The WITNESS list file is generated in SFMS by rules which are matched against CRMS and whose Right Hand Side (RHS) actions generate WITNESS list file statements under the ***define*** and ***detail*** sections.

The format of the list file generation rules is:

```
rule:
        [LHS Matches CRMS]
->
        [RHS actions for writing WITNESS list file statements]
```

The actual rules, and their application, for transferring part and machine details into a WITNESS list file are presented in Appendix C.

5.4.1.2 Translator for PROMODEL

Structure of PROMODEL

PROMODEL uses different tables within a single text file for internal representation of the model. The different tables are used for representing the information entered in the Routing, Part Scheduling, Downtime, Capacity, Transporter Path, etc. modules. The Part Scheduling table, as an example, has the layout:

Part	Location	Qty per arrival	No of arrivals	Start(min)	Arrival Frequency(Min)

Translation Methodology for PROMODEL

The PROMODEL tables are generated by rules which are matched against CRMS data facts and whose RHS actions generate entries in the various tables. The modelling of parts in PROMODEL involves generating entries in the routing and part scheduling tables.

The format of the rules used for writing the entries is of the form:

```
rule:
        [LHS Matches CRMS]
->
        [RHS actions for writing PROMODEL table entries]
```

The actual rules, and their application, for translating part and machine data can be found in

Appendix C

5.4.1.3 Translator for FACTOR/AIM

Structure of FACTOR/AIM

FACTOR/AIM uses tables for representing a model in the DB2 database management system, but is different from PROMODEL in that separate tables(not within the same file) are used to represent different parts of the model. Examples of some of the commonly used tables and their descriptions are given in Fig 5.21.

Table	**Description**
ORDERnnn	This table defines the type of order. There are three different types of orders including: a ***new*** order released at a specific time; an ***in-process*** order to represent a order released into the system prior to the start of the simulation; an ***explicit-release*** order to release an order into the system whenever a release jobstep is performed.
DEMANDnnn	This table is used to define characteristics of orders. It includes:

	expected makespan; first release time; inter-arrival time; maximum to release; name; number of parts per load, etc.
PROCPLNnnn	This table defines the process plan of an order. It contains: the name of the first jobstep; the list of remaining job-steps
JOBSTEPnnn	This defines the individualjobsteps of a particular process plan
RESRCnnn	Single-capacity resource table for representing a the characteristics of a single machine. Characteristics include: allocation type; animation type; its statistics display options (e.g. the average time it was down); options for collecting queue length data; specification of shift patterns.
MCRnnn	Multi-capacity resource table for representing the characteristics of a multi-capacity machine. The characteristics specified are the same as for a single capacity resource with the addittion of a column for ***capacity***.
POOLnnn	General and WIP buffer tables. The characteristics specified are the same as for multi-capacity resource.
CONSYSnnn	Conveyor System table. The characteristics specified include: type; block rule; down rule; spacing, etc.
CONSEGnnn	Conveyor Segment table defines a portion of conveyor system. The characteristics specified include: capacity; begin and end control points; name; length; etc.
CONCPnnn	Conveyor Control Points table defines where pickups and drop-offs occur.
TRNSYSnnn	Transporter system table defines name and description of a

	transporter system.
TRNSEGnnn	Defines segments of transporter layout using start and end control points
TRNCPnnn	Transporter control points where pickups and drop offs are performed

Fig 5.21 The main FACTOR/AIM database tables

The JOBSTEPnnn table, as an example, has the following layout:

DESCR	JSID	NEXTJSID	PROCPLANID
Load arrives to system	in_sys	lb01	l-bracket
take pre-cast parts to drill	lb01	lb02	l-bracket

where:

- DESCR is the jobstep description.
- JSID is the jobstep identification.
- NEXTJSID is the next jobstep identification.
- PROCPLANID is the process plan accessed by thejobsteps.

Translation Methodology for FACTOR/AIM

The FACTOR/AIM tables are generated by rules which are matched against CRMS data , whose RHS actions generate different table entries. In FACTOR/AIM, parts are modelled using the tables ORDERnnn, DEMANDnnn, PROCPLnnn and JOBSTEPnnn. The generation of the DEMANDnnn and JOBSTEPnnn are illustrated here; most of the part characteristics are transferred into these tables.

The format of the rules used for generating the tables is of the form:

```
rule:
        [LHS Matches CRMS]
->
        [RHS actions for writing FACTOR/AIM table entries]
```

The actual rules, and their application, for translating part and machine data can be found in Appendix C

5.4.1.4 Natural language description

In addition to the application reference models or translators for (currently three) manufacturing simulators, SFMS translation module also contains a translator to transform the instanstiated data in CRMS into a natural language description of the model specification. This is designed to help the user in the verification of the specification and its documentation.

Verification is important when dialogues are used since, if there are a substantial number of sequential dialogues, it is virtually impossible for the user to keep track of the information he has specified. A natural language documentation is also likely to be useful when the need arises for translation of a model into a different simulator and/or the merger of parts of models oroginally written in different simulators.

The generation of a natural language description involves translation from one language to another. Traditionally research has been geared to translating one spoken language like German (source language) into another, say, English (the target language). This involves first translating the source language into a meaning representation, requiring thorough knowledge of the language and the following steps:

1. tokenisation to analyse individual words into their components, with separation from punctuation.
2. structural analysis to transform sequences of words into structures, rejecting those word sequences that breach the grammar rules of the language.
3. discourse analysis to determine how the meaning of a sentence may be influenced by the previous sentence.
4. pragmatic analysis to determine what is meant. For example "do you know what day it is" should be answered with a "particular day" and not simply "yes".

The main problem with language translation is that generation of a meaning representation requires world knowledge for discourse and pragmatic analysis, which is beyond the current limitations of AI. This can only be overcome by placing restrictions on the sentences that are to be analysed. After a meaning representation has been generated, the translation into the target language is less knowledge intensive (only structural knowledge of the target language is required), since a subset of grammatical rules can be used to generate a limited number of target language statements.

For our purpose, which is translation of the information entered through the dialogues into a English language description, the analysis of the dialogue input is restricted and requires no structural, discourse, or pragmatic analysis. This is because the input is restricted to fixed elements based on domain knowledge. The meaning representation is the data in CRMS, and is generated from variables instantiated from the user inputs through dialogues. Then grammatical rules are used to translate the information in CRMS into an English language description

For example a rule that transforms the inter-arrival information in CRMS for part 1 below

```
frame(creation_info,1,inter_arr_tim,val,triangular_distribution),
frame(creation_info,1,p1,val,2),
frame(creation_info,1,p2,val,4),
frame(creation_info,1,p3,val,6)]
```

is

```
rule 1#:
[frame(creation_info,X,inter_arr_tim,val,triangular_distribution),
frame(creation_info,X,p1,val,B),
frame(creation_info,X,p2,val,C),
frame(creation_info,X,p3,val,D)]
=>
[retract(all),
tell('engspec.pl'),                          %open file for output of
                                               English specification
write('The interarrival time of part '),
write(X),                                    %writes part number
write(' is according to a triangular distribution with a '),nl,
write('minimum of  '),
write(B),                                    %write minimum value
write(' seconds, a mode of '),
write(C),                                    %write mode value
write( ' seconds and a maximum of '),
write(D),                                    %write maximum value
write(' seconds.'),nl].
```

and generates the sentence:

```
The interarrival time of part 1 is according to a triangular distribution with a
minimum of 2 seconds, a mode of 4 seconds and a maximum of 6 seconds.'
```

Similarly the following two rules generate sentences describing the lot sizes and first arrival times of parts.

```
rule 16#:
[frame(creation_info,X,lot_size,val,B)]
=>
[retract(all),
tell('engspec.pl'),
write('The lot size for part '),
write(X),                                    %write part number
write(' is '),
write(B),write('.')].                        %write lot size
```

```
rule 17#:
frame(creation_info,X,first_arr_tim,val,B)]
=>[
```

```
[retract(all),
tell('engspec.pl'),
write('The first arrival  for part '),
write(X),                                    %writes part number
write(' is at time '),
write(B),write('.')].                        %writes arrival time of 1st part
```

5.5 Behavioural Knowledge

Knowledge discussed so far is concerned with the structural elements of manufacturing systems, and has been the main focus of SFMS. The problem with including behavioural knowledge (apart from the process plan) is the lack of any standard, with different systems having different means of including such knowledge: for example, FACTOR/AIM has a number of different in-built rules with no means of tailoring, whilst WITNESS and PROMODEL have a number of different distinct elemental rules which can be used alone or combined to form more complex composite rules.

However, behavioural knowledge for restricted domains can be included in a standard framework. In jobshop and batch production systems, for example a number of simple sequencing rules are commonly used to direct resource allocation and material flow; examples of such rules are First Come First Served (FCFS), Earliest Due Date (EDD), Shortest Processing Time (SPT), etc. In FACTOR/AIM these traditional priority or dispatching rules are directly available as options.

In other simulators , the priority rules have to be made up of elemental rules that the system provides. For example, consider the case where the contents of a buffer are to be sequenced for subsequent processing by a machine on the basis of the SPT rule, i.e according to the shortest or minimum processing time. This can be implemented in WITNESS using: attribute ***pull_this*** (for identifying part with minimum processing time); integer variable ***position*** (position in buffer); real variable ***min_prot*** (minimum processing time); and real variable ***pro_tim*** (processing time***).***

A function ***minprt*** used in the action on entering the buffer, shown below, can then be used for finding the part in the buffer with the minimum processing time.

```
NAME OF FUNCTION: minprt;
TYPE: Name;
PARAMETERS: l
        buf,Name
ACTIONS, Execute
Add
   min_prot = buf:r_cy_m6              %sets initial value of min_prot
   FOR l = 1 TO NPARTS (buf)           %scans each part in turn
       pro_tim = buf AT l:r_cy_m6      %sets pro_tim to cycle time of part at position l
       IF pro_tim <= min_prot %performs test to see if parts cycle less than minimum
           pro_tim = min_prot              %If so min_prot set to parts cycle time
           position = l              %sets position to l in buffer to identify part so far with
       ENDIF                           minimum cycle time
   NEXT                                %repeats for remainder of parts
   buf AT position:pull_this = 1 %sets attribute pull_this of part with min cycle time
   RETURN buf                          at position.
End Actions

END minprt
```

the buffer, detailed below, then pushes the part which has the maximum value (which is 1) of attribute ***pull_this*** to the connected machine.

```
NAME OF BUFFER:buffer1;
QUANTITY: 1;
CAPACITY: 1000;
DELAY TIME : 0.0;
INPUT POSITION: Rear;
OUTPUT SCAN FROM: Front;
Select: Maximum;          %output part whose attribute pull_this is
Of: pull_this;                        maximum (i.e equals 1)
REPORTING: Individual;
SHIFT: Undefined,0;

END buffer1
```

This could be implemented in a equivalent PROMODEL representation using the ***IF-THEN*** rule in the ***operation*** column, as shown below in Fig 5.22. In the example below, at the location ***Buffer*** the action in the ***operation*** column checks to see if attribute At3 (processing times at m1) of part types p1and p2 against attribute At4 (minimum processing time at m1). The part type which has the minimum processing time is then sent to m1 using the action ***SEND***.

Part	Location	Operation(Min)	Output Part	Next Location	Condition	Qty	Move time(Min)
p1	Buffer	If AT3=AT4 then SEND 1 p1 TO M1	p1	M1	If	10	0
p2	Buffer	If AT3=AT4 then SEND 1 p1 TO M1	p2	M1	If	10	0
p1	m1	10	p1	unload	0	1	0

Fig 5.22 SPT rule in PROMODEL

5.6 Linking SMSF with Simulators

The data model in the Standard Manufacturing Simulation Framework (SMSF) provides a standard representation for the development/maintenance of the definition of the main structural and process interaction (process route) elements of manufacturing systems, and their characteristics, usually found in generic simulators. It has also been shown that for relatively restrictive application domains, for example as envisaged in FACTOR/AIM, much of the commonly used priority and assignment rules for governing the behaviour of system elements can also be incorporated in such a standard data model; elemental rules found within simulators can also be included, but not composite rules which are often created by the user from them. The application reference models contained in the translation module of SMSF can then be used to produce a basic or partial system model in the internal representations of target simulators. Hence, if it were to be adopted as the standard for all new and existing simulators, SMSF would greatly assist in the transfer of models between different simulators; only non-standard, mainly behavioural, elements would need to be added in the target simulator once the transfer was completed.

For such a framework to work effectively, the standard data model should ideally be completely integrated within all simulators. The user interface in SMSF (or one similar to it) should become part of all new or existing simulators, and the (standard) internal data representation (CRMS) of the

framework should be maintained by them in parallel with their own. This would ensure that the definition of the common elements and their characteristics are consistently maintained in CRMS for subsequent translation, when necessary. The approach still provides the vendors of individual simulators plenty of scope for product differentiation.

If, however, the framework is not truly integrated with a simulator, then the user must exercise strict discipline in ensuring that the common elements and their characteristics are defined, and maintained only through SMSF; otherwise consistency of definition between the two internal data representations will be lost. In such a situation, it may be helpful to have reverse engineering tools that would transfer model data from the target simulators to CRMS data. Development of such reverse engineering tools, however, poses a much more challenging problem since the internal data representation of the simulator would first need to be parsed to determine common system elements (and their characteristics) and their meanings, translation can then take the same rule based approach as already used in SFMS. If true standardisation is to be achieved, then vendors would be expected to incorporate that standard data model and an appropriate translator as part of their product, including the necessary reverse engineering tool with it.

Chapter 6. Conclusions

The research effort furnished the following conclusions:

1. The major motivation in simulation research has been concerned with improving the task of model specification; this has always been considered important both to reduce the considerable time and effort needed for it, and to widen its use by reducing the level of expertise required.

2. Early research in simulation was concerned with the development of world views on which modelling approaches could be standardised. Three such world views emerged in the 1960's, namely event scheduling, activity scanning and process interaction. The activity scanning approach, from the perspective of the manufacturing domain, has a machine dominant view of the world (as does the event based approach), and is particularly well suited for modelling production lines (mass production systems). Activity scanning provides the best modular structure for model specification but computationally the approach is relatively inefficient. Process interaction, from the perspective of the manufacturing domain, has a material dominant view of the world, with process routes of parts as the central building block for specification of the real world system. The approach is best suited for manufacturing systems with a large variety of parts and routings, as is usually the case in intermittent production systems (batch production, jobshops), and the world view is similar to that which production engineers take of such systems.

3. The very earliest simulation systems were usually toolkits which provided users with a set of routines that are commonly required in simulation modelling. The toolkits use an event based view of the world and, although the model specification still had to be provided in 3rd generation computer languages, the task of the modeller was eased by the availability of standard routines. Soon a number of new languages dedicated to simulation modelling became available. Most of these were based on the activity and process interaction based views of the world (e.g Hocus for the former, and GPPS for the latter), although a few were based on the event scheduling approach (SIMSCRIPT).
4. The next major development in easing the task of simulation modelling came from the application of automatic programming techniques, which raised the specification level from the simulation language code to a higher more natural level using questionniares, dialogue interfaces, natural language interfaces, graphical interfaces, etc. Early systems were simply code generators while later research led to the development of automatic modelling systems which incorporated domain specific knowledge. The influence and actual use of AI tools and techniques in automatic simulation programming systems has been considerable, particularly in the later systems. This can be found in the three main areas:-improved interfaces for model specification (e.g in the use of dialogues and natural language; better schemes for representing simulation, domain and target language knowledge (e.g. use of frames, object oriented techniques); and use of knowledge based methods for translating the model specification into a

model in the target language. However, due to the logic intensive nature of simulation models their application does have limitations. The restriction lies not only in the domain of application for automatic modelling systems, but also in the limitations in modelling complex behaviour of real world systems; often the user would need to extend or modify the model created by enhancement to the code in the underlying simulation language or, alternatively, simply accept a certain loss of accuracy in the model.

5. The development of simulators:

a)meant early systems were highly restrictive in their application domain (e.g FMS), and were completely data-driven. This made them very easy to use, but totally inflexible in modelling behaviour that was not already incorporated within the particular system. The more recent generic manufacturing simulators, the subject matter of this research, extended the application domain to more general manufacturing systems. They removed many of the problems of modelling inflexibility not only by enhancing the inherent domain knowledge but also by providing the user with a built-in programming language to add complex behavioural aspects of the real world system, and/or the ability to drop out into an external computer language to do the same.

b) was made possible by advances in computing techniques, as well as AI concepts of knowledge based systems.

c) resulted in the incorporation of manufacturing domain knowledge in the specification language used to elicit knowledge of the real world system from the user and, thus, eased the task of model specification.

d) resulted in the use of modelling constructs to raise the specification level from that of abstract concepts of entities, queues and events, activities or processes to that of real world objects like machines, parts, labour and process routes.

e) emphasised differences in their world view, with some exhibiting a machine based world view, others a material based view of the world, while some (e.g. WITNESS) have developed a hybrid view incorporating both alternatives; the particular world view determines whether they are naturally more suited to modelling certain manufacturing systems as discussed earlier. However, they also to some extent incorporate features from the activity and process interaction based approaches; for example the way interrupting activities (machine breakdowns, setup, etc) are modelled has a distinctive activity based view, while the time advancement mechanism usually is similar to that for the process interaction approach.

6.1 Research achievements and contribution

Many different manufacturing simulators are now commercially available, and their number is rapidly increasing. Lack of any common standard is posing a growing problem. It is difficult to move a model developed in one simulator to another, or to combine models developed in different systems. When such becomes necessary or is considered desirable, not only does a modeller have to learn the syntax and vocabulary of a new language, a considerable time and

effort would be needed in rewriting the entire model. Development of a standard is necessary to improve reusability of models written in different systems and is the first step, in the interchangability of such models; this was the motivation behind the research project.

1. The research has lead to the development of a Standard Manufacturing Simulation Framework (SMSF). It has used automatic programming techniques, and advances in knowledge elicitation, representation and methods, and consists of a standard data model and a translation module for conversion of the specification data internal representations of target simulators. The framework has been developed in LPA Prolog.
2. The standard data model has been developed based on an analysis of generic manufacturing simulators; in particular three popular systems (WITNESS, PROMODEL and FACTOR/AIM) were analysed in detail, which were specifically chosen for the differences in their modelling approach and data representation methods. The analysis revealed a high degree of commonality of modelling elements, and their characteristics, used in the simulators to represent the structural component of manufacturing simulators.
3. The case of behavioural components defining interactions between resources, priority rules for allocation of resources, etc, is, however, very different; the simulators have widely different methods for expressing such logic. The only behavioural element for which a standard could be easily defined is a process route providing basic information on the interaction

between parts and resources (machines, transporters), and this has been included in the standard framework. Methods for inclusion in the framework of priority rules for allocation of resources to jobs (e.g. EDD, SPT, etc) however, have been indicated. These rules are directly available as options in FACTOR/AIM, but have to be translated into other simulators using the programming and elemental modelling constructs they provide.

4. The standard data model uses a dialogue interface to elicit from the user the model specification data; this has been implemented using LPA Prolog's dialogue manager. A Common Representation for Manufacturing Simulators (CRMS) has been developed for internal model data representation. CRMS has been implemented using the frame AI knowledge representation scheme. It provides a highly modular and readable mechanism for storage of instantiated model, and is the basis for the exchange of model data. Each frame represents a common element (module), and the resulting modularity promotes compatibility with the internal model representations of manufacturing simulators. The standard data model is consistent in simulators which have a material dominant view of the world or a machine dominant view, or a hybrid view as in WITNESS. However, it captures more of the data for a simulator with a material dominant view or hybrid view through a process route.

5. A production rule based method has been used to provide the translation methodology in the standard framework, SMSF, and this has been illustrated with the development of translators for the three popular simulators that have been studied in detail. Their developments are based

on an analysis of the structure of the internal model representation of the target simulators, determining the equivalent representation of frames of CRMS which are compatible with these structures, and the design of production rules which would transfer model data from CRMS to an equivalent internal model structure of the target simulators. The translators contain specifications of the format of the data structures of CRMS and the internal model representation of the generic simulators, together with a data transfer mechanism for mapping constructs syntactically from the former to the latter. The modularity of CRMS and its compatibility with the internal model representations of the generic manufacturing simulators made the design and coding of the rules within the translators relatively simple, concerned solely with data mapping; although the translators are application specific (i.e. dependent on the target simulator, the methodology used is common and generally applicable).

6. The research has addressed the real problem of transferring model data between generic manufacturing simulators. SMSF provides a first step towards the development of standards in the field, and the structural integration of model data of dissimilar simulators. The maintenance of a minimum model specification through the standard data model will reduce substantially the effort needed in transferring a simulation model from one simulator to another.

If the standards that have been developed are accepted by the vendors and users of simulators, there are two ways for application of the framework. Ideally it would be integrated by the vendors in their systems, with the standard data model an integral part of the model specification language and internal model data representation; this would ensure automatic consistency of data. Any new simulator that is developed should use the standard data model as the blue print. There will still be plenty of scope for product differentiation through enhancements to modelling capability, the method offered for representing behavioural aspects of the real world system, the way the simulation executive works, and the reports that are generated. The alternative method would be to maintain the standard data model in parallel with that of the simulator that is currently in use by the model developers. In such cases, strict discipline would be required to ensure that the common modelling elements and their common characteristics are maintained through the standard data model, as otherwise consistency of data between the two systems would be lost.

6.2 Limitations of framework.

The limitations of the framework are:

1. It only covers only the structural aspects of manufacturing systems with the only allowance for the behavioral aspects being the process plan.

2. The translators only generate partial WITNESS, PROMODEL and FACTOR/AIM models. Although a sufficient number of rules for each translator have been written to provide a basis for their completion.

6.3 Further Work

A number of suggestions for further work and enhancements to the standard framework have been made in the thesis, and three areas need special mention.

1. ***Analysis of other simulators.*** Only three simulators have been analysed in-depth in the study, although others have been considered by studying their brochures and published literature. A more in-depth analysis of other manufacturing simulators, e.g. XCELL+, SIMFACTORY, ARENA, MODEL MASTER, TAYLOR II, etc, would help to further validate the standards that have been developed (and their refinement, if necessary). Translators for them could also be developed.
2. ***Greater analysis and inclusion of behavioural elements.*** Currently the standard framework includes very little representation of behavioural elements of a real world system and, hence, is only capable of producing partial models. Indications, however, have been provided on how priority rules common used for assignment of resources (e.g. EDD, SPT). Greater analysis of such behavioural elements would be needed. This would have two great advantages. First, this should mean that fully working models

would be produced during translation (which would normally still require enhancements or modifications). Secondly, in some cases a complete automatic translation may be possible, e.g. in the case of simple models, and models in very specific domains (batch production, jobshops which use standard priority rules as found in FACTOR/AIM for instance).

3. ***Development of a reverse engineering tool.*** As already discussed, if the standard data model is not fully integrated with the specification language and internal data representation of manufacturing simulators, then strict discipline is required to maintain consistency of data between the systems. In such cases, the availability of a reverse engineering tool that would translate the data from the internal representation of a simulator to that of the standard framework would be of great benefit. Development of such reverse engineering tools would require careful formalised parsing of the internal data representations of the simulators and analyses of their meaning. This is a considerable research challenge but, if successful, it would close the loop in the framework for model translation and interchangability.

References

AESOP. 1994. SIMPLE++ Reference manual, AESOP GmbH, Konigstrabe 82, D-70173, Stuttgart.

Adelsberger, H.H. 1984. *Prolog as a simulation language.* Proceedings of the 1984 Winter Simulation Conference. (Nov.): 501-504.

Adelsberger, H.H.; and G. Neumann. 1985. *Goal Oriented Simulation Modelling using Prolog.* Proceedings of the 1985 SCS Conference on Modelling and Simulation on Micro-Computers, San Diego, CA, USA. (Jan.): 42-47.

Adelsberger, H.H.; U.W. Pooch; R.E. Shannon; and G.N. Williams. 1986. *Rule Based Object Oriented Simulation Systems.* Intelligent Simulation Systems, SCS Simulation Series, 17, No. 1. (Jan): 107-112.

Arrons,H.De. Swann. 1983. *Expert Systems in the Simulation Domain.* Mathematics and Computers in Simulation, Vol XXV.

Backus, J.W; and H. Herrick. 1954. *IBM 701 speed coding and other automatic programming systems.* Proceedings of the Symposium of Automatic Programming for Digital Computers, Office of Naval Research, Washington D.C.

Balmer, D.W.; and R.J. Paul. 1986. *CASM-The right environment for simulation.* J. Opl Res. Soc. Vol. 37: 443-452.

Balzer, R.M.; N. Goldman; and D. Wile. 1976. *On the transformational approach to programming.* Proceedings of the 2nd International Conference on Software Engineering: 337-344.

Balzer, R.M. 1981. *Transformational Implementation: An Example.* IEEE Transactions on Software Engineering, vol SE-7, no. 1(Jan.): 3-13.

Balzer, K.; and T.E. Cheatham; C. Green. 1983. *Software Technology in the 1990's: Using a New Paradigm.* Computer 16, no. 11 (Nov.): 39-45.

Balzer, R.M. 1985. *A 15-Year Perspective on Automatic Programming.* IEEE Trans. Software Eng., Vol. 11, no. 11(Nov.): 1257-1267.

Banks, J; E. Aviles; J.R. McLaughlin; and R.C. Yuan. 1991.
The Simulator: New Member of the Simulation Family. Interfaces 21:2 March-April:76-86.

Barr, A.; and E.A. Feigenbaum (Eds.). 1981.
The Handbook of Artificial Intelligence, Vol. 1, 1981 and Vol. 2, 1982. William Kaufman, Los Altos, Calif.

Barstow, D. 1977.
A Knowledge Based System for automatic program construction. IJCAI 5: 382-388.

Barstow, D. 1979.
Knowledge Based Program Construction. Elsevier, Amsterdam.

Barstow, D. 1984.
A Perspective on Automatic Programming. AI Magazine, Vol. 5, no. 1(Spring.):5-27.

Barstow, D. 1985.
Domain Specific Automatic Programming. IEEE Trans. Software Eng., Vol. 11, no. 11(Nov.): 1321-1336.

Baskaran, V.; and Y.V. Reddy. 1984.
An Introspective Environment for Knowledge Based Simulation. Proceedings of the 1984 Winter Simulation Conference, Dallas, Tex: 645-651.

Bengu. G, and J.Haddock. 1986.
A Generative Simulation-Optimisation System Journal of Computers & Industrial Engineering, 1986, Vol.10, No.4, pp.301-313

Bevans, J.P. 1982.
First, Choose an FMS Simulator. American Machinist, vol 126, no5 (May):143-145.

Bhattacharyya, R.Roy, and Y.S.Huang. 1989
Knowledge Bases Techniques for Control of FMS. Proc. Bejing. Int. Conf. in Simulation and Scientific Computing.

Biermann, A.W. 1975.
Approaches to Automatic Programming. Advances in Computers 15, Academic Press, New York.

Biermann, A.W.; and B.W. Ballard. 1980.
Toward natural language computation. American Journal of Computational Linguistics, Vol. 6, no. 2 (Apr-Jun.): 71-86.

Biggerstaff, T.J. 1976.
A super-compiler approach to automatic programming. Phd thesis, Technical Report 76-01-01, Computer Science Department, University of Washington.

Black, W.J. 1986.
Intelligent Knowledge Based Systems, An Introduction. Van Nostrand Reinhold (UK) Co. Ltd.

Bobrow, D.G. 1985. *The Software Engineering of Expert Systems: Is Prolog Appropriate ?*.IEEE Trans. Software Eng., Vol. 11, no. 11 (Nov.): 1391-1399.

Boehm, B.W.; and T.A. Standish. 1983. *Software Technology in the 1990's: Using a Evolutionary Paradigm.* Computer 16, no.11 (Nov.): 30-37.

Boling, B.A; M.A Layman. 1986. *ModelMaster Factory Modeling System Tutorial.* Proceedings of the 1986 Winter Simulation Conference. J Wilson and S.Roberts (eds).

Bratko, I. 1986. Prolog Programming for Artificial Intelligence. Addison-Wesley Publ Ltd.

Brazier, M.K.; and R.E. Shannon. 1987. *Automatic Programming Of AGV Simulation Models.* Proceedings of the 1987 Winter Simulation Conference, Atlanta, GA, USA: 16-24.

Burkett, W.C; Y.Yang. 1992 *On the implementation of STEP,* ISO TC184 SC4/WG4 N68.

Buxton J. N; and J.G. Laski. 1962 *Control and Simulation Language.* The Computer Journal 5.

Campbell, J.A. 1984 Implementations of Prolog. Ellis Horwood, Chichester, England.

Carrie, A.S. 1986. *The Role of Simulation in FMS.* In Flexible Manufacturing Systems: Methods and Studies. A.Kusiak (Ed.). North Holland, Amsterdam, New York.

Carrie, A.S. 1988. Simulation of Manufacturing Systems. John Wiley and Sons.

Carson, J.S. 1986. *Convincing Users of a Model's Validity is the Challenging Aspect of Modeller's Job.* Industrial Engineering (Jun.).

Cheatham, T.E. 1984. *Reusability Through Program Transformation.* IEEE Transactions on Software Engineering, Vol. SE-11, no. 11 (Nov.): 1296-1320.

Clark,K.; and S. Sickel. 1977. *Predicate logic: A calulus for deriving programs.* IJCAI 5: 419-420.

Cleary, J.G.; and A. Dewar. 1984. *Interpreters for logic programming-a powerful tool for simulation.* Proceedings of the SCS Conference on Simulation in Strongly Typed Languages, San Diego, CA, USA (Feb.).

Cleary, J.G.; K.S. Goh; and B. Unger. 1985. *Discrete Event Simulation in Prolog.* Proceedings of the SCS Conference on Artificial Intelligence, Graphics and Simulation, San Diego, CA, USA (Jan.):8-13.

Clementson, A.T. 1982. Extended Control and Simulation Language, CLE Coms Ltd, Birmingham.

Clocksin, M.; and C. Mellish. 1981. Programming in Prolog. Springer-Verlag:Berlin.

Conway. R; W.L. Maxwell. 1986. *XCELL+:A Cellular, Graphical Factory Modeling System.* Proceedings of the 1986 Winter Simulation Conference. J Wilson and S.Roberts (eds).

Crookes, J.G.; D.W. Balmer; S.T. Chew; and R.J. Paul. 1986. *A three-phase simulation system written in Pascal.* J. Opl Res. Soc. Vol. 37: 603-618.

DaSilva, J.M.Bastos. 1986. *The Use of Decision Mechanisms in Visual Simulation for Manufacturing Systems Modelling.* AI Applied to Simulation, v18, n1, SCS:165-170.

Dahl, O.J;and Nygaard, K. 1966. *SIMULA-an ALGOL-based Simulation Language.* CACM 9 (9):671-678.

Dar-El, E.M; and Wysk, R.A. 1982. *Job Shop scheduling-A sytematic approach.* Journal of Manufacturing Systems, 1 (1), 77-88.

Darlington, J.; and R.M. Burstall. 1973. *A system which automatically improves programs.* IJCAI 3: 479-485.

Davies, N.R. 1976. *A Modular Interactive System for Discrete Event Simulation Modelling.* Proceedings of the 9th International Conference in Systems Simulation, Hawaii (Jan.): 296-299.

Davies, N.R. 1979. *Interactive Simulation Program Generation.* Methodology in Systems Modelling and Simulation, North Holland Publishing Company Ltd: 179-200.

Davis, C.G.; and C.R. Vick. 1977. *The Software Development System.* IEEE transactions in Software Engineering Vol. SE-3, no. 1 (Jan.):69-84.

Doshi, K.A.; S. Madala; and J.B. Sinclair. 1985. *GIST a Tool for Specifying Extended Queuing Network Models.* Technical Report TR8511. Rice University, Department of Electrical and Computer Eng, Houston, Tex (May.).

Duersch, R.R.; and M.A. Laymon. 1985. *Programming Free Graphic Factory Simulation with GEFMS/PC..* General Electric Company, Corporate Research and Development, Schenectady, N.Y.

Elmaraghy, H.A. 1982. *Simulation and graphical animation of advanced Manufacturing Systems.*Journal of Manufacturing Systems, 1 (1), 53-63.

Elmaghraby, A.S.; and V. Jagannathan. 1986. *An Expert System for Simulationists.* Proceedings on the Conference on Intelligent Simulation Environments (P.A. Luker and H.H. Adelsberger, eds), SCS Simulation Series Vol. 17, no. 1, San Diego, CA: 42-47.

Endesfelder, and Templemeier. 1987. *SIMAN Module Processor-A Flexible Tool for Generating Siman Simulation Models.* Applied Informatics, vol 29, no 3 (Mar):104-110.

Evans, J.B. 1984. *Simulation and Intelligence.* Computer Studies Publications No. TR-A5-84, Center of Computer Studies and Applications, University of Hong Kong.

Fickas, S.F. 1985. *Automating the transformational development of software.* IEEE Transactions on Software Engineering, Vol. SE-11, no. 11 (Nov.): 1268-1277.

Fiddy, E; J.G. Bright; and R.D. Hurrion. 1981. *SEE-WHY interactive simulation on the screen.* Institute of Mech Eng Conference Publications: pp 167-172.

Fishman, G.S. 1978. Principles of Discrete Event Simulation. John Wiley and Sons Inc.

Fishwick, P.A. 1988. *Automating the transition from lumped models to base models.* Proceedings of the Conference on AI and Simulation, Orlando, FL, USA (18-21 Apr.): 57-63.

Flitman, A.M.; and R.W. Hurrion. 1987. *Linking discrete-event simulation models with expert systems.* J. Opl Res. Soc. Vol. 38, no. 8: 723-734.

Frankowski, E.N.; and W.R. Franta. 1980. *A process oriented simulation and model specification documentation language.* Software-Practice and Experience 10: 721-742.

Ford, R.; and J.B. Schroer. 1987. *An Expert Manufacturing Simulation System*. Simulation 48, no.5: 193-200.

Fox, M.S. 1982. *The Intelligent Management System: An Overview*. Report No CMV-RI-TR-81-4. Carnegie-Mellon University, Pittsburgh, Pa (Dec.).

Fox , M.S; N.Sathi; V Baskaran; and J Bouer. 1986. Simulation Craft:An Expert System for Discrete Event Simulation. In Simulation III:Proceedings of the SCS Simulation Conference. The Society for Computer Simulation, San Diego CA.

Frenkel, K.A. 1985. *Toward Automating the Software Development Cycle.* Communications Of the ACM 28, no.6 (June.): 578-589.

Futo, I.; and J. Szeredi. 1982. *A Discrete Simulation System based on Artificial Intelligence Techniques*. In Discrete Simulation and Related Fields, A.Javor (Ed.), North Holland, Amsterdam: 135-150.

Gaines, B.R.; and M.L.G. Shaw. 1985, *Expert Systems and Simulation*. In Proceedings of the 1985 Conference on Artificial Intelligence, Graphics, and Simulation. San Diego, Calif (Jan.): 95-101.

Gong, D.C;and L.F, McGinniss. 1990. *An AGVs Simulation Code Generator for Manufacturing Applications*. Winter Simulation Conference Proceedings (Dec): pp 676-682.

Gordon, G. 1961. *A general purpose simulation system.* In Proceedings Eastern Joint Conference, Washington D.C, Dec 12-14, Macmillan, New York: 87-104.

Ginsberg, A.S.; H.M. Markowitz; and P.M. Oldfather. 1965. *Programming by Questionnaire.* Rand Memorandum RM-4460-PR, The Rand Corporation, Santa Monica, CA (Apr.).

Green, C. 1969. *The application of theorm proving to question-answering systems.* Phd thesis, AIM-96, Electrical Engineering Department, Stanford University, USA.

Green, C.; R.P. Gabriel; B.I. Kedzierski; J.V. Phillips; S.T. Tappel; and S.J. Westfold. 1979. *Results in Knowledge Based Program Synthesis.* E. Kant; Proceedings of the 6th International Conference on Artificial Intelligence, Tokyo, Japan (20-23 Aug.): 342-344.

Greenwood, N.R. 1988. Implementing Flexible Manufacturing Systems. Macmillan Education Ltd, Hampshire RG21 2XS.

Haddock, J.; and R.P. Davis. 1985. *Building a Simulation Generator for Manufacturing Cell Design and Control*. Proceedings of the 1985 IIE Spring Conference, Los Angeles (May.).

Haddock, J. 1987. *An expert system based on a simulation generator*. Simulation 48, no. 2 (Feb.) : 45-53.

Haddock, J. 1988. *A Simulation Generator for Flexible Manufacturing Systems Design and Control*. IIE transactions Vol. 20, no. 1 (Mar.): 22-31.

Haddock, J.; and R.M. O'Keefe. 1990. *Using Artificial Intelligence to facilitate Manufacturing Systems Simulation*. Computers Ind. Engng. Vol. 18, no. 3: 275-283.

Haider, S.W.; and J. Banks. 1986. *Simulation Software Products for Analyzing Manufacturing Systems*. Industrial Engineering (Jul.).

Hayes, R.F.; D.A. Waterman; and D.B. Lenat. 1983. Building Expert Systems, Addison-Wesley, Reading, Ma.

Heidorn, G.E. 1974. *English as a very high level language for simulation programming*. SIGPLAN Notices 9, no. 4, 91-100.

Heidorn, G.E. 1976. *Automatic programming through natural language dialog: A survey*. IBM J. Research and Development, Vol. 20: 302-313.

Henriksen, J.O.; and R.C. Crain. 1983. *Overview of the GPSS Language*. GPSS/H User's Manual, 2nd Edition, Wolverine Software Corporation, Annandale VA.

Hills. 1965 *SIMON-Simulation language in Algol*. Simulation in Operational Research, English University Press, London.

ICI. 1990 PROPHET sales literature, ICI CSMD, Runcorn.

Insight Ltd, 1993. INORDA sales literature, Insight Logistics Ltd, The Quadrangle, Woodstock, Oxon, OX20 1LH.

Istel Ltd, 1993. Provisa User's Manual, AT&T Istel Ltd, Highfield House, Headless Cross Drive, Redditch, Worcs, B97 5EQ.

Istel Ltd, 1995. WITNESS User's Manual, AT&T Istel Ltd, Highfield House, Headless Cross Drive, Redditch, Worcs, B97 5EQ.

Khoshnevis, B.; and A.P. Chen. 1986. *An expert simulation model builder.* Proceedings on the Conference on Intelligent Simulation Environments (P.A. Luker and H.H. Adelsberger, eds), SCS Simulation Series Vol. 17, no. 1, San Diego, CA: 42-47.

Kiran , A.S; and M.L. Smith. 1983. *Simulation Studies in Jobshop Scheduling: A Survey.* Proc of the Conference on Simulation in Inventory and Production Control, San Diego, California. Soc for Computer Simulation:46-51.

Kiviat; and Colker. 1964. *GASP-A GENERAL ACTIVITY SIMULATION PROGRAM,* The Rand Corporation, Santa Monica, California.

Klahr, P.; W.S. Faught; and G.R. Martins. 1980. *Rule-Oriented Simulation.* In Proceedings of the International Conference on Cybernetics and Society. IEEE, Boston, Mass (8-10 Oct.): 350-354.

Kowalski, R. 1977. *Predicate logic as a programming language.* North Holland, Amsterdam.

Law A.M; and W.D.Kelton. 1982 Simulation modelling and analysis. McGraw Hill Book Company, NY.

Law A.M. 1983. *Statistical Analysis of Simulation Output Data.* Operations Research, 31:983-1029.

Lehmann, A.; B. Knodler; E. Kwee; and H. Szczerbicka. 1985. *Dialog-oriented and knowledge-based modelling in a typical pc environment.* Proceedings on the Conference on Intelligent Simulation Environments (P.A. Luker and H.H.Adelsberger, eds), SCS Simulation Series Vol. 17, no. 1, San Diego, CA: 133-138.

Lenat, D.B. 1975. *Synthesis of large programs from specific dialogues.* In Proving and Improving Programs (G. Huet and G. Kahn, eds), Institute de Recherche d'Informatique et d'Automatique.

Lenz, J.E; and J.J. Talavalage. 1977. *Generalised Computerised Manufacturing Systems Simulator.* The Optimal Planning of Computerised Manufacturing Systems, Report Number 7, Purdue University, Indiana, USA.

Lenz, J.E. 1988 *The MAST Simulation Environment for analysing low inventory systems.* Proceedings of the 1988 Winter Simulation Conference. (Eds M,Abrams, P.Haigh, and J.Comfort):194-197.

Lenz, J.E. 1988 Flexible Manufacturing:Benefits for the low inventory factory. Marcel Dekker Inc, New York.

Lenz, J.E. 1988. The Mast User Manual, CMS Research Inc, Oshkosh, Wisconsin, USA.

Lenz, J.E. 1989 *The MAST Simulation Environment.* Proceedings of the 1989 Winter Simulation Conference. (Eds E.A MacNair, K.J. Musselman, P.Heidelberger):243-248.

Love, P.L.; and P.M. Oldfather. 1968. *Programming by Questionnaire: Auxiliary Programs.* Rand Memorandum RM-5689-PR, The Rand Corporation, Santa Monica, Ca (Aug.).

Malpas, J. 1987. Prolog: A relational language and its applications. Prentice Hall Ltd, Englewood Cliffs, NJ.

Manna, Z.; and R. Waldinger. 1980. *A deductive approach to program synthesis.* ACM Trans. Programming Languages and Systems, Vol. 2, no.1 (Jan.): 90-121.

Markowitz, H.M; R Hauser; and H. W Karr, 1963 *SIMSCRIPT-A simulation programming language,* Report from the Rand Corporation, Santa Monica, California.

Martin, W.A. 1974. *A system for building expert problem solving systems involving verbal reasoning.* OWL notes, Project Mac, Massachusetts Institute of Technology.

Mathewson, S.C.; and J.A. Allen. 1978. *DRAFT/GRASP.* Proceedings of the Tenth Annual Simulation Symposium, Tampa.

Mathewson, S.C. 1982. *DRAFT II/SIMON Manual.* Proceedings of the Department of Management Science, Imperial College, London.

Mathewson, S.C. 1983. *User Acceptance:Design Considerations for a Program Generator.* Software-Practice and Experience, Vol. 13, no. 2: 101-117.

Mathewson, S.C. 1984. *The Application of Program Generator Software and Its Extension to Discrete Event Simulation Modelling.* IIE Transactions, Vol. 16, no. 1 (March.).

Mathewson, S.C. 1985. *Simulation Program Generators: code and animation on a PC.* J. Opl Res. Soc. Vol. 36: 583-589.

Mcarthur, D.; and H. Sowizral. 1981 *An object oriented language for constructing simulations.* Proceedings of the 7th International Conference on Artificial Intelligence, Vancouver, Canada (Aug.): 809-814.

Mccune, B.P. 1977. *The PSI Program Model Builder-Synthesis of very high level programs.* SIGART-SIGPLAN Symposium on Artificial Intelligence, Rochester, NY, USA (15-17 Aug.): 130-139.

McDougall, M.H. 1987. Simulating Computer Systems: Techniques and tools, MIT press.

Mellichamp, J.M.; and A.F.A. Whab. 1987. *An expert system for FMS design.* Simulation 48, no. 5 (May.): 201-208.

Merritt, D. 1989. Building Expert Systems in Prolog. Springer-Verlag, New York NY10010 USA. (Eds, S.S. Muchnik, and P. Schnupp).

Mier, S.R.; J. Talavage; and D. Ben-Arieh. 1985. *Towards a Knowledge-Based Network Simulation Environment.* Proceedings of the 1985 Winter Simulation Conference, San Francisco, CA (11-13 Dec.): 232-236.

Mills R; and J.J.Talavage. 1985. *Simulation Programs for FMS design.* Proceedings of the 1st International Conference on Simulation in Manufacturing (ed W.Heginbottom): 217-229.

Minsky. 1979. *A Framwork for Representing Knowledge.* In Frame Conceptions and Text Understanding (D. Metzing, ed.):1-25.

Montazeri, M; L.f. Gelders; and L.N.Van Wassenhove. 1988. *The Modular Simulator for Design, Planning and Control of Flexible Manufacturing Systems.* The International Journal of Advanced Manufacturing Technology, 3 (1):15-32.

Morgan, B.J.T. 1984. Elements of Simulation. Chapman and Hall, 733 Third Avenue, New York NY10017.

Murray, K.J.B. 1986. *Knowledge-based model construction: An automatic programming approach to simulation modelling.* dissertation, Dept of Computer Science, Texas A & M University, College Station, Tex (Dec.)

Murray, K.J.B.; and S. Sheppard. 1986. *Design of a Knowledge Based Model Construction System: The Integration of Simulation Modelling, Automatic programming and Expert systems.* Technical Report TAMUDCS-86-001.Texas A & M University, Department of Computer Science, College Station, Tex (Jan.).

Musselman, K.J. 1984. *Computer Simulation: A Design Tool for FMS.* Manufacturing Engineering (Sep.): 117-120.

Nordgren, B. 1994. *Taylor II manufacturing simulation software.* In 1994 Winter Simulation Conference Proceedings, eds J.D. Tew, etal: 446-449. Association for Computing Machinery, New York, N.Y.

Novels, D; K.E. Wichmann, 1990. *Simulation Applied To Production Scheduling.* 2nd International Conference on Factory 2000- Integrating Information and Material Flow, Cambridge, U.K.(July).

Nyan, P.A. 1987. *A comprehensive environment for object oriented simulation of manufacturing systems.* Simulation in Computer Integrated Manufacturing, K.E. Wichmann (Ed.), European Simulation Multiconference, SCS Publications: 1-25.

Nygaard. 1978. *Development of the SIMULA languages.* ACM SIGPLAN notices vol 13, no 8: 245-273.

O'Keefe, R. 1986. *Simulation and Expert Systems -A Taxonomy and Some Examples.* Simulation 46, no. 1 (Jan.): 10-16.

O'Keefe, R.; and R.M. Davis. 1986. A *microcomputer system for simulation modelling.* Eur. J. Opl Res. Vol. 24, no. 1: 23-29.

O'Keefe, R.; and J.W. Roach. 1987. *Artificial Intelligence approaches to Simulation.* J. Opl Res. Soc. Vol. 38, no. 8: 713-722.

O'Keefe and Haddock. 1991. *Data-Driven Generic Simulators For Flexible Manufacturing Systems.* International Journal Of Production Research, vol 29, no 9:1795-1810.

Oldfather, P.M.; A.S. Ginsberg; H.M. Markowitz. 1966 *Programming by Questionnaire: How to Construct a Program Generator.* Rand Memorandum RM-5129-PR, The Rand Corporation, Santa Monica, CA (Nov.).

Oldfather, P.M.; A.S. Ginsberg; P.L. Love; and H.M. Markowitz. 1967. *Programming by Questionnaire: The Job Shop Simulation Program Generator.* Rand Memorandum RM-5162-PR, The Rand Corporation, Santa Monica, CA (July).

Oren, T.I. 1977. *Simulation -As it has been, is, and should be.* Simulation 29, no.5 (Nov.): 182-183.

Oren, T.I. 1982. *Computer-Aided Modelling Systems.* In Progress in Modelling and Simulation (F.E. Cellier, ed.). Academic Press, New York, N.Y.: 189-203.

O'Shea, T.; J.Self; and G. Thomas. 1987. Intelligent knowledge based systems :An introduction. Harper & Row, London.

Partsch, H.; and T. Steinbruggen. 1983. *Program Transformation Systems.* IEEE Transactions on Software Engineering, vol SE-11, no. 11 (Nov.): 1268-1277.

Paul, R.J.; and G.I. Doukidis. 1986. *Further developments in the use of artificial intelligence techniques which formulate simulation problems.* J. Opl Res. Soc, Vol.36, No.12, p.1170.

Paul, R.J.; and S.T. Chew. 1987. *Simulation modelling using a interactive Simulation Program Generator.* J. Opl Res. Soc., Vol. 38: 443-452.

Pegden, C.Dennis. 1985. Introduction to SIMAN. Systems Modelling Corp, State College, PA.

Pidd, M. 1988. Computer Simulation in Management Science, John Wiley, New York.

Pritsker 1972. *Q-GERT: GERT networks with queueing capabilities.* RM 72-7, technical report, Purdue University. Dec.

Pritsker 1974. The GASP-IV simulation language, John Wiley, New York.

Pritsker and Pegden. 1979. Introduction to Simulation and SLAM, John Wiley and Sons, New York.

Pritsker Corp. 1972. FACTOR/AIM users guide, 1910 Purdue Road, Indianapolis, IN 46268, USA.

Quillian 1968 *Semantic Memory.* In Semantic Information Processing, eds M.Minsky, MIT Press, Cambridge, MA.

Ranky, P.G. 1983. The Design and Operation of FMS. IFS (Publications) Ltd, U.K.

Raczynshi, S. 1990. *Graphical Description and Program Generator for Queuing Models.* Simulation (Sept): 147-153.

Reddy, Y.V.; and M.S. Fox. 1982. *KBS (Knowledge Based Simulation): An Artificial Intelligence Approach to Flexible Simulation.* Technical Report CMU-RI-TR-82-1. Carnegie-Mellon University, Intelligent Systems Laboratory, The Robotics Institute, Pittsburgh, Pa (14 Sep).

Reddy, Y.V.; and M.S. Fox. 1982. *Knowledge representation as in organisational modelling and simulation:A detailed example.* Proceedings of the 13th Annual Pittsburgh Conference on Modelling and Simulation. (April.).

Reddy, Y.V.; M.S. Fox; and N. Husain. 1985. *Automating the Analysis of Simulations in KBS.* In Proceedings of the 1985 Conference on Artificial Intelligence, Graphics and Simulation (G.Birtwistle,ed.). SCS. San Diego, Calif (Jan.): 34-40.

Reddy, Y.V.; M. Fox; N. Husain; and M. Roberts. 1986. *The Knowledge Based Simulation System.* IEEE Software 3, no. 2 (March): 26-37.

Rich, C.; and R.C. Waters. 1978. Readings in Artificial Intelligence and Software Engineering. Morgan Kaufman, Los Altos, California.

Rich, C.; and R.C. Waters. 1988. *Automatic Programming:Myths and Prospects.* Computer (Aug.): 40-50.

Ringland and Duce. 1988. Approaches to Knowledge Representation:An Introduction. Eds G.A. Rigland and D.A. Duce. Research Studies Press Ltd.

Reilly, K.D.; W.T. Jones; and P. Dey. 1985. *The Simulation Environment Concept Artificial Intelligence Perspective.* Proceedings of the 1985 Eastern Simulation Conferences:Artificial Intelligence and Simulation (W.M. Holmes, ed.) SCS, Norfolk, Va (3-8Mar.): 29-34.

Robertson, P. 1985. *A Rule Based Expert Simulation Environment.* Proceedings on the Conference on Intelligent Simulation Environments (P.A. Luker and H.H. Adelsberger, eds), SCS Simulation Series Vol. 17, no. 1, San Diego, CA: 9-15.

Rolsten, L.J. 1985. *Modeling Flexible Manufacturing Systems with MAP/1.* Annals of Operations Research 3:189-204.

Roussel, P. 1975. PROLOG:Manuel de Reference et d'utilisation, University of Aix-Marseilles, Luminy, France.

Rummel, P.A 1988. *Simtalk:Pros and cons of natural language for manufacturing.* Proceedings of the 1st International Conference on Industrial Applications of AI and Expert Systems. vol 12:97-21.

Russel, E.C. 1983. Building simulation models with SIMSCRIPT II.5, CACI-Federal, Los Angeles, California.

Sabuncuoglu, I.; and D. Hommertzheim. 1988. *Computer usage in manufacturing systems: Expert Systems and Simulation In Flexible Manufacturing Systems.* Computers Ind. Eng. Vol. 15, nos 1-4: 1-7.

Sabuncuoglu, I.; and. D. Hommertzheim. 1989. *Expert Simulation Systems-Recent developments and applications in Flexible Manufacturing Systems.* Computers Ind. Eng. Vol 15, nos 1-4: 1-7.

Sammett, J.E. 1966. *The use of English as a programming language.* Comm ACM (March):228-229.

Schank, R.C; R.P Abelson. 1977. Scripts, Plans, Goals and understanding, Erlbaum, Hillsdale, N.J.

Schnupp, P.; C.T. N. Huu.; and L.W. Bernhard. 1989. Expert Systems Lab Course. (Ed, S.S. Muchnik). Springer-Verlag Berlin Heidelberg, New York.

Schriber, T.J. 1987. *The Nature and Role of Simulation in the Design of Manufacturing Systems.* Graduate School of Business Administration. The University of Michigan, Ann Arbor, Michigan 48 109-1234 USA.

Schroer, B.J.; and F.T. Tseng. 1988. *Modelling complex manufacturing systems using discrete event simulation.* Comput Ind. Eng., Vol. 14, no.4: 455-464.

Schroer, B.J. 1989. *A Simulation Assistant for modeling Manufacturing Systems.* Simulation (Nov):201-205.

Schwetman, H. 1986. *Using CSIM to model complex systems.* In the 1988 Wnter Simulation Conference, San Diego, CA:246-253.

Selvaraj, S; E.Blair; M.L. Smith; and W.M.Marcy. 1990. *Discrete Event Simulation in C with DISC.* Computers ind Engng Vol 18, No 3:263-274.

Shanehchi, J. 1985. *EXPRESS:A man-machine interface for simulation.* Proceedings of the 1st International Conference on Simulation in Manufacturing (Ed W.B. Heginbotham). IFS Publications ltd, Bedford, UK (March):85-97.

Shannon, R.E. 1975. *AI based simulation environments.* Intelligent Simulation Environments, SCS Simulation Series, Vol. 17, no. 1 (Jan.): 150-156.

Shannon, R.E.; R. Mayer; and H.H. Adelsberger. 1985. *Expert systems and simulation.* Simulation 44, no. 6 (Jun.):275-284.

Shannon, R.E.; R. Mayer; and D.T. Phillips. 1986. *Knowledge based simulation techniques for manufacturing.* ULTRATECH Conference Proceedings, Vol. 1, Section 1, SME Paper MS 86-966, (Sep.):237-262.

Shannon, R.E. 1988. *Knowledge based simulation techniques for manufacturing.* Int. J. Prod. Res., Vol. 26, no. 5: 953-973.

Shapiro, S.C. 1987. Encylopaedia of Artificial Intelligence. Wiley:New York.

Shaw, M.L.G.; and B.R. Gaines. 1986. *A Framework for Knowledge-Based Systems Unifying Expert Systems and Simulation.* In Proceedings of the Conference on Intelligent Simulation Environments (P.A. Luker and H.H. Adelsberger, eds.). SCS Simulation Series 17, no. 1, San Diego, Calif. (23-25 Jan.): 38-43.

Shearn, D.C.S. 1990 *PASSIM-A Pascal discrete event simulation program generator.* Simulation (July):31-38.

Sinclair, J.B.; K.A. Doshi; and S. Madala. 1985. *Computer Performance Evaluation with GIST: A Tool forSpecifying Extended Queuing Network Models.* Proceedings of the Conference on Intelligent Simulation Environments, SCS Simulation Series, Vol. 17, no. 1, San Diego, CA (11-13 Dec.): 290-299.

Sinclair, J.B.; and S. Madala. 1986. *A Graphical Interface for pecification of Extended Queuing Network Models.* Proceedings of the 1986 Fall Joint Computer Conference, Dallas, Tex (2-6 Nov.)

Sinclair, J.B.; and S. Madala. 1986. *A Graphical Interface For Specification of Extended Queuing Network Models.* In Proceedings of the 1986 Fall Joint Computer Conference. ACM, IEEE-CS. Dallas, Tex (2-6 Nov.): 709-718.

Smith, D.R.; G.B. Kotik; and S.J. Westfold. 1985. *Research on Knowledge-Based Software Environments at Kestrel Institute.* IEEE Transactions on Software Engineering, Vol. SE-11, no. 11 (Nov.): 1278-1295.

Strandhagen, J.O. 1987 — *Expert knowledge in object oriented simulation of manufacturing systems.* Proceedings, 3rd Int Conf on Simulation in Manufacturing. NTH-SINTEF, Norwegian Inst of Technology, Norway (Nov.): 23-30.

Sterling, L.; and Shapiro, E. 1987. — The Art of Prolog. MIT Press. Cambridge, MA.

Subrahmanian, E.; and R.L. Cannon. 1981. — *A Generator Program for Models of Discrete Event Systems.* Simulation 36, no. 3 (Mar.): 93-101.

Swartz, J.T. 1975. — On Programming:An interim report on the SETL Project, Revised. Computer Science Department. New York University (June).

Swartout, B. 1982. — *Gist English Generator.* Proceedings of the National Conference on Artificial Intelligence AAAI-82.

Tokoro, M., Ishikawa, Y. 1984. — *An object oriented approach to knowledge systems.* Proc. International Conference on Fifth Generation Computer Systems, Tokoyo, Japan, Nov 6-9. Institute of of New Generation Computer Technology: 623-631.

Trapp, G. 1991 — *Concurrent Engineering PDES/STEP,* Product Data Exchange for the 1990s-Seminar Proceeding, New Orleans, Louisiana.

Waldinger, R.; and K.N. Levitt. 1974. — *Reasoning about programs.* Artificial Intelligence 5:235-316.

Warren,D.H.; L.M. Pereria;and F.C.N Pereria. 1977. — *PROLOG- The language and its implementation compared to LISP.* SIGPLAN Notices 12-8.

Waters, R.C. 1985. — *The Programmers Apprentice: A Session with KBEmacs.* IEEE Trans. Software Eng., Vol. SE-11, no. 11 (NOv.): 1296-1320.

Wyvill, B. 1985 — *Current trends in graphics and simulation.* In Proceedings of the 1985 Conference on Artificial Intelligence, Graphics, and Simulation. San Diego, Calif (Jan.): 95-101.

Y.Yang. 1993 — *The STEP Integration Information Architecture.* Engineering Data Management: Key to Success in a Global Market.

Zalevsky, P.A. 1988. — *Knowledge based simulation of manufacturing facilities.* Proc of the SCS Multi-conference on Artificial Intelligence and Simulation: The diversity of applications (Feb.): 67-71.

Zaniolo, C. 1984. *Object Oriented Programming in PROLOG.* Proceedings of the International Symposium on Logic Programming, Atlantic City, N.J. Feb 6-9, IEEE Computer Society, New York: 265-271.

Zeigler, B.P. 1987. *Knowledge representation from Newton to Minsky and beyond.* Applications of Artificial Intelligence, Vol. 1, no. 1:87-107.

Appendix A WITNESS Detail Forms, PROMODEL Modules and FACTOR/AIM Modelling Editors

The WITNESS detail forms, PROMODEL modules and the FACTOR/AIM modelling editors are given in this section and are referenced from chapter 4 section 4.4.

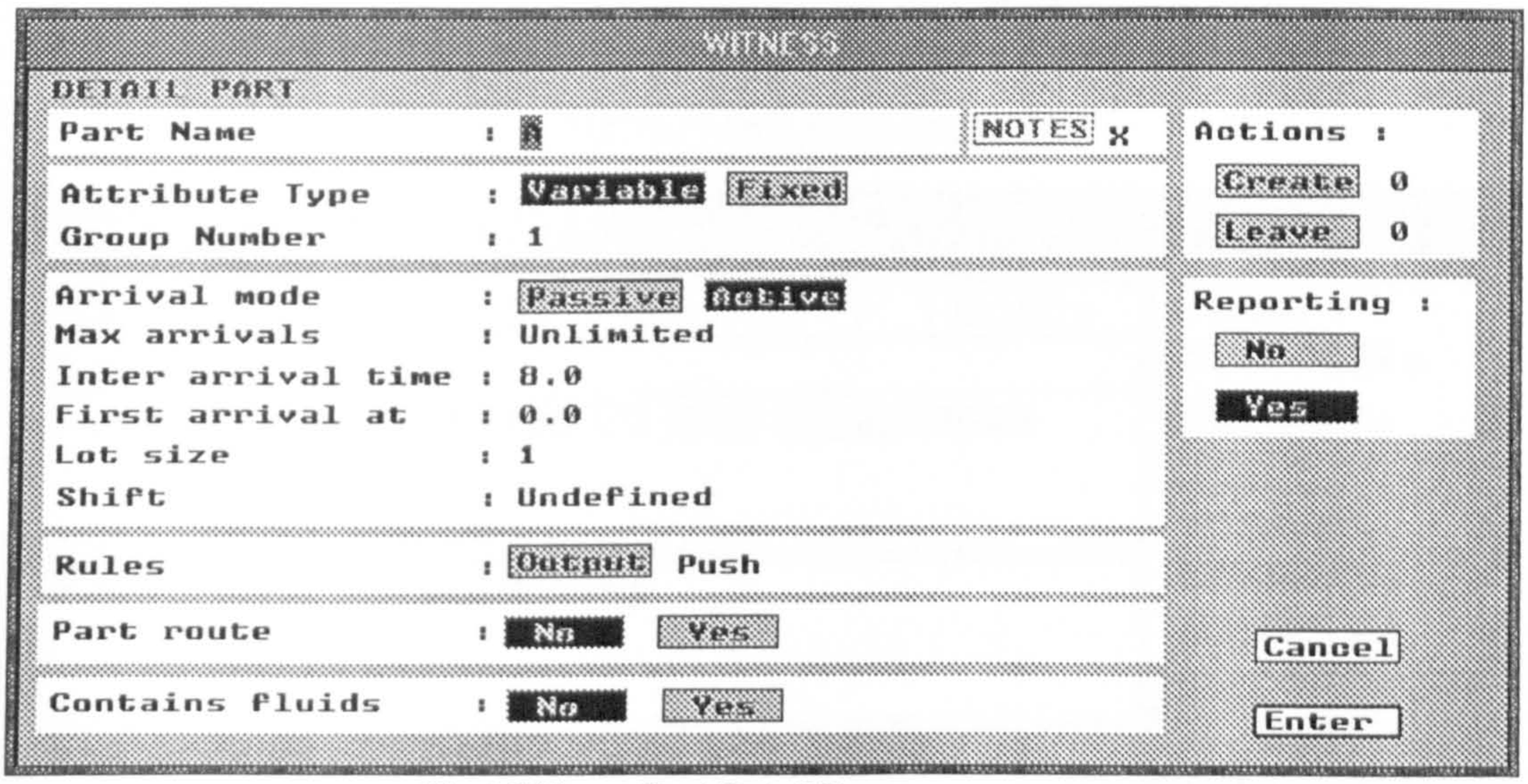


Fig 1 Witness Part Detail Form

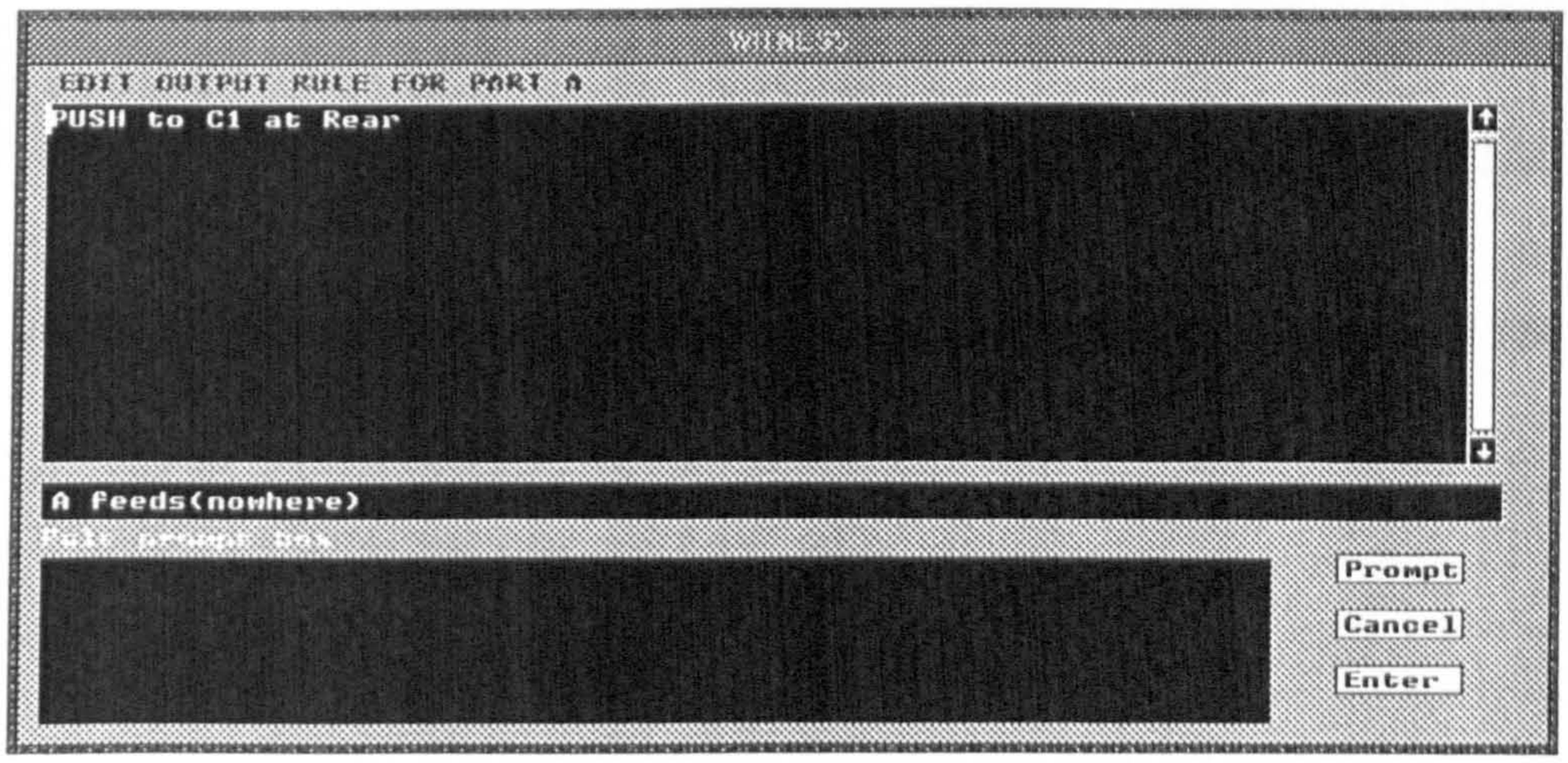


Fig 2 WITNESS Part Output Rule Editor

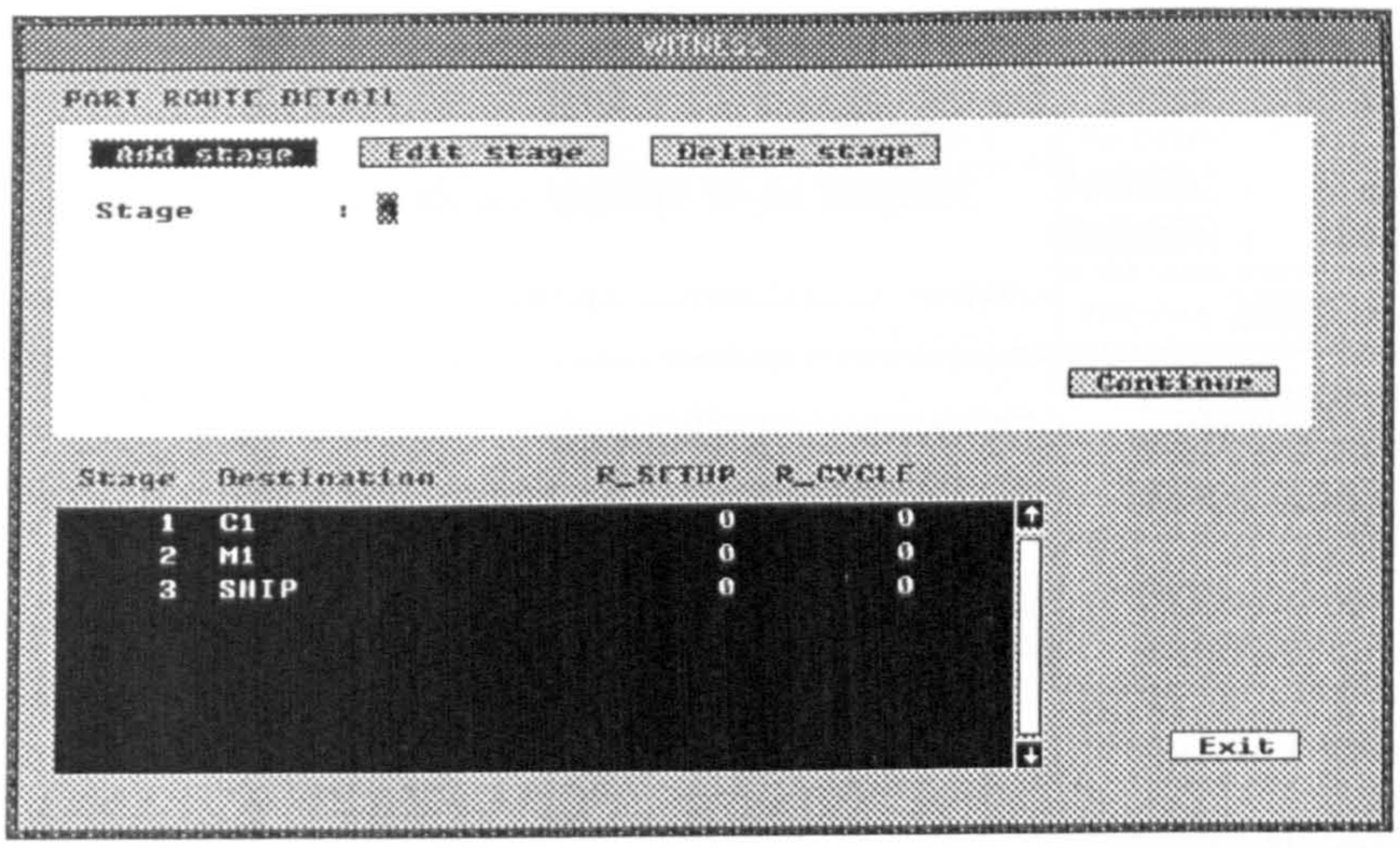


Fig 3 WITNESS Part Route Editor

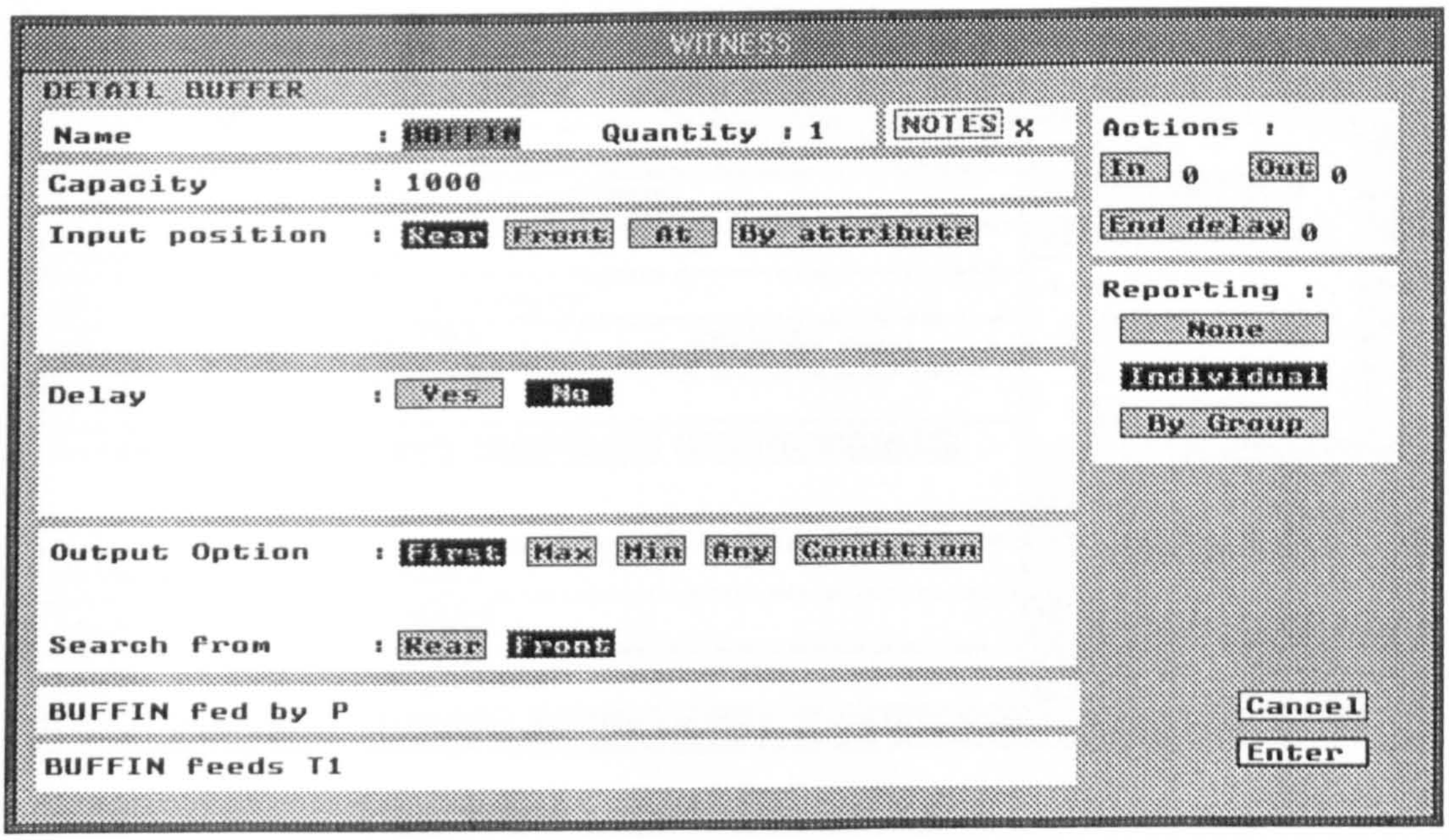


Fig 4 WITNESS Buffer Detail Form

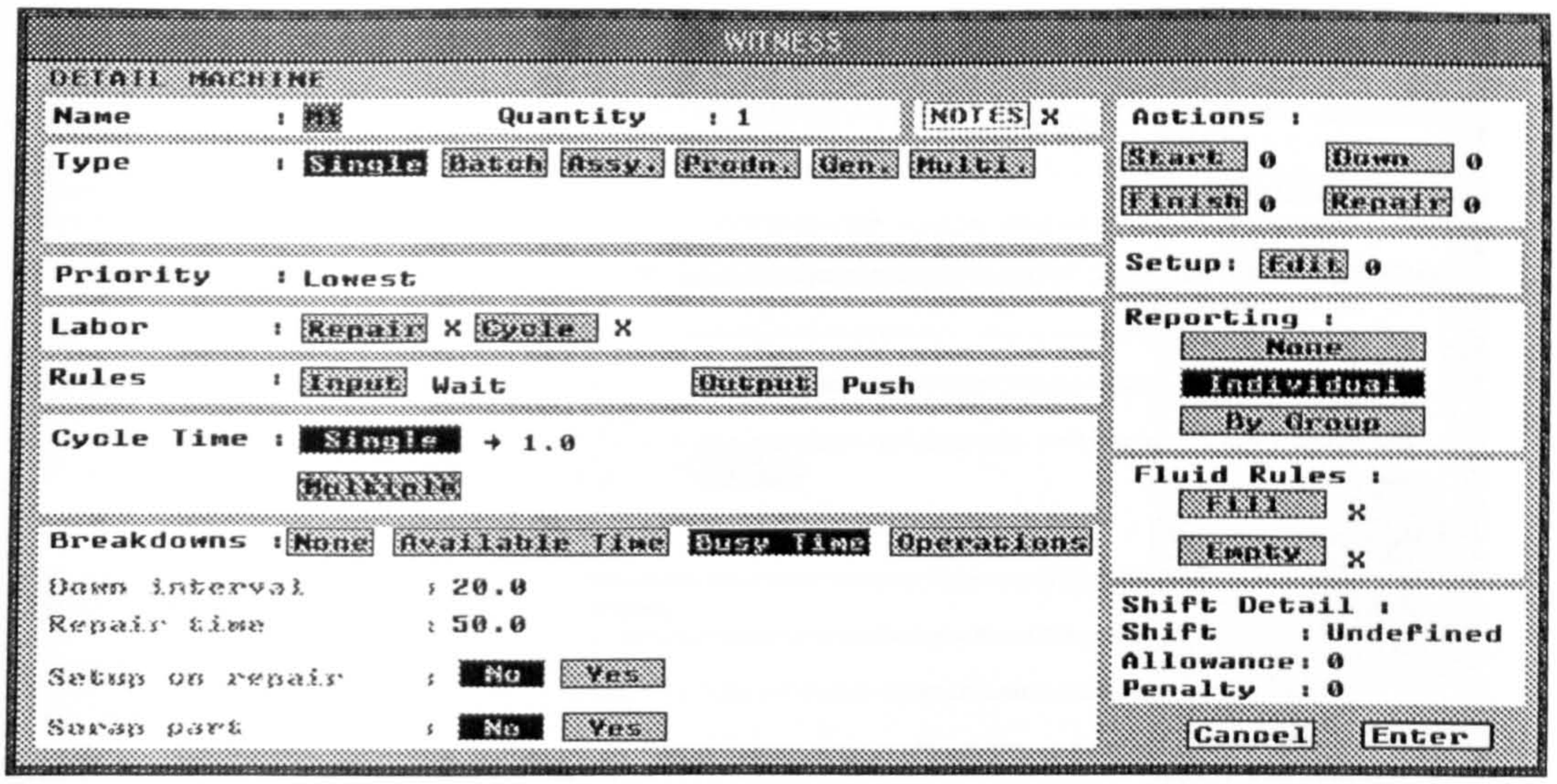


Fig 5 WITNESS Machine Detail Form

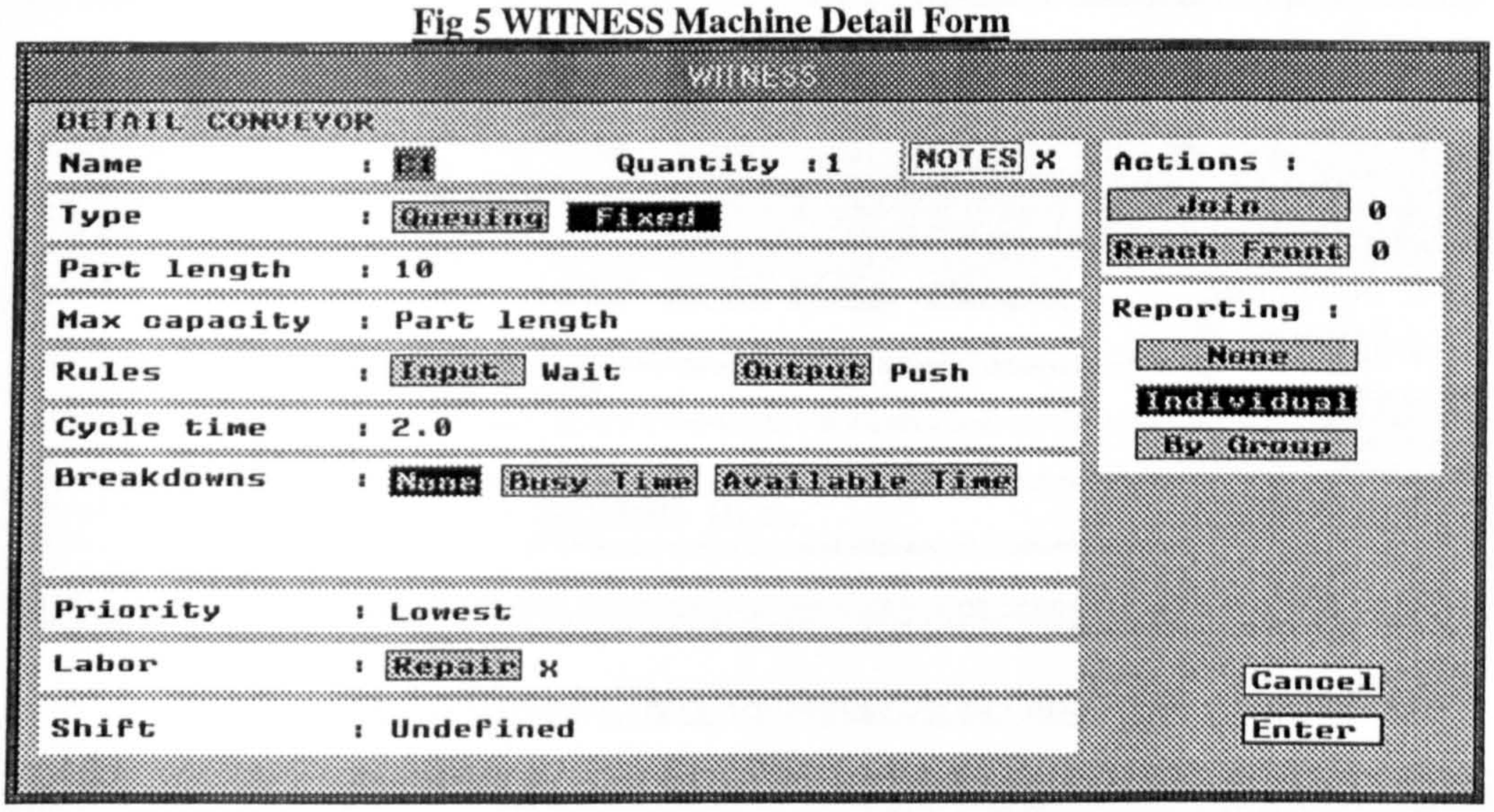


Fig 6 WITNESS Conveyor Detail Form

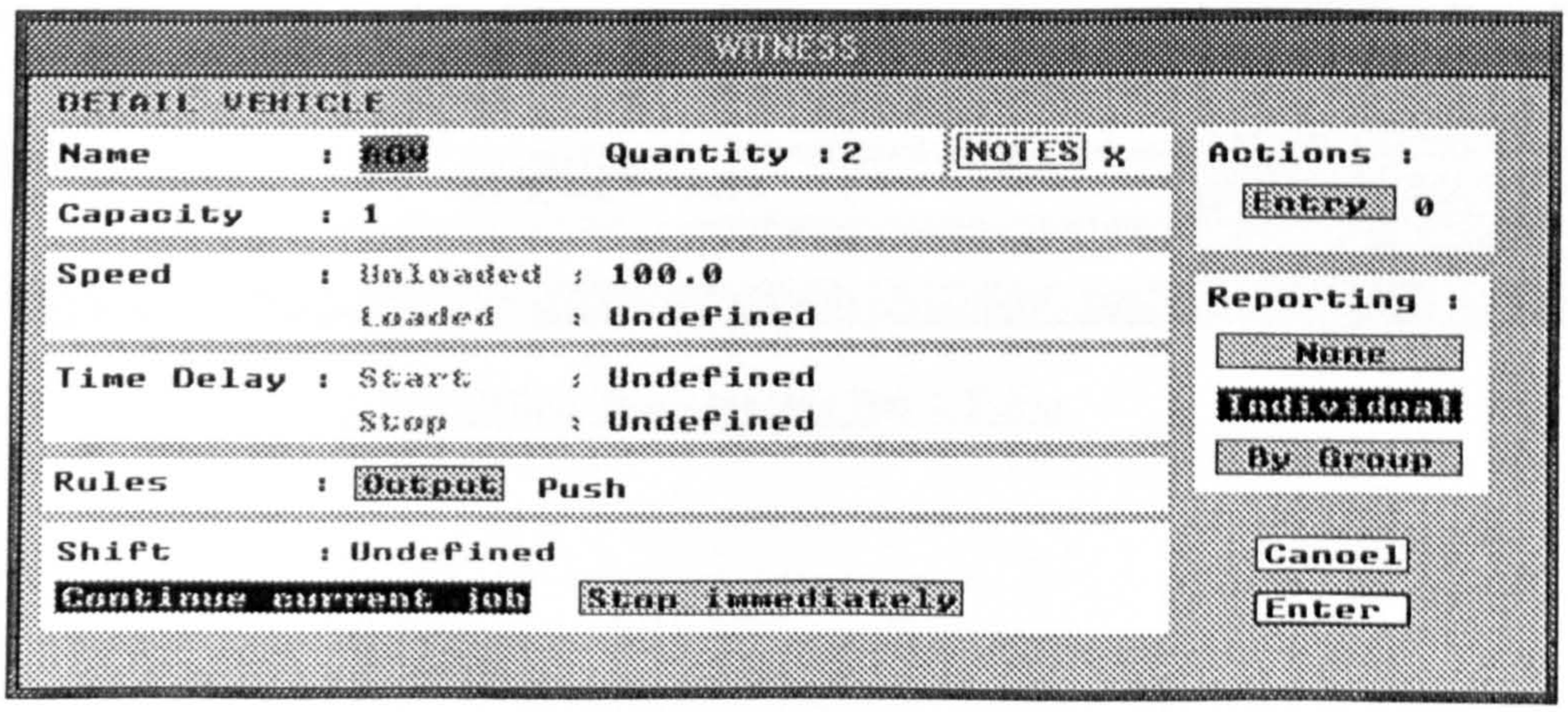


Fig 7 WITNESS Vehicle Detail Form

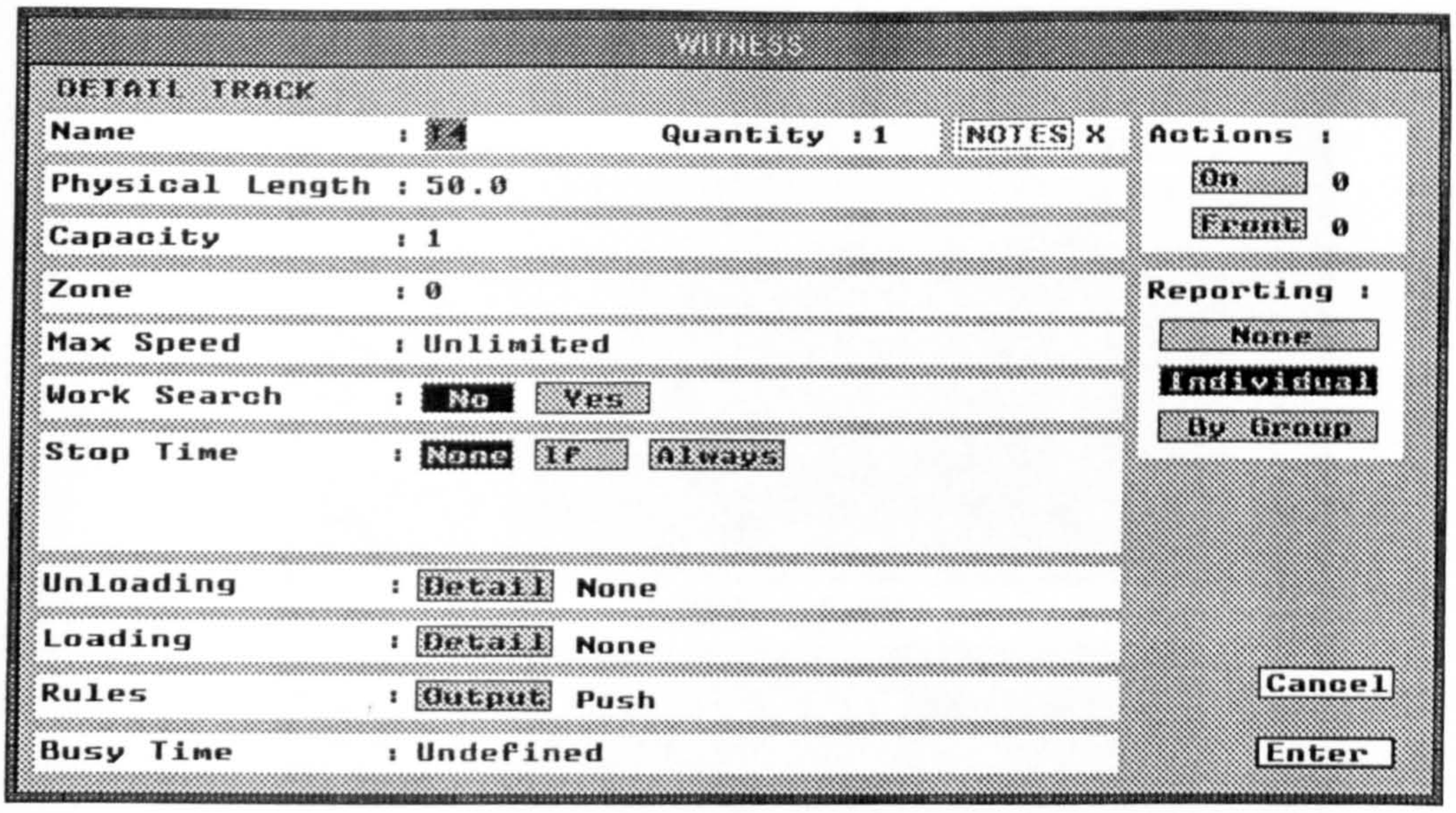


Fig 8 WITNESS Track detail Form

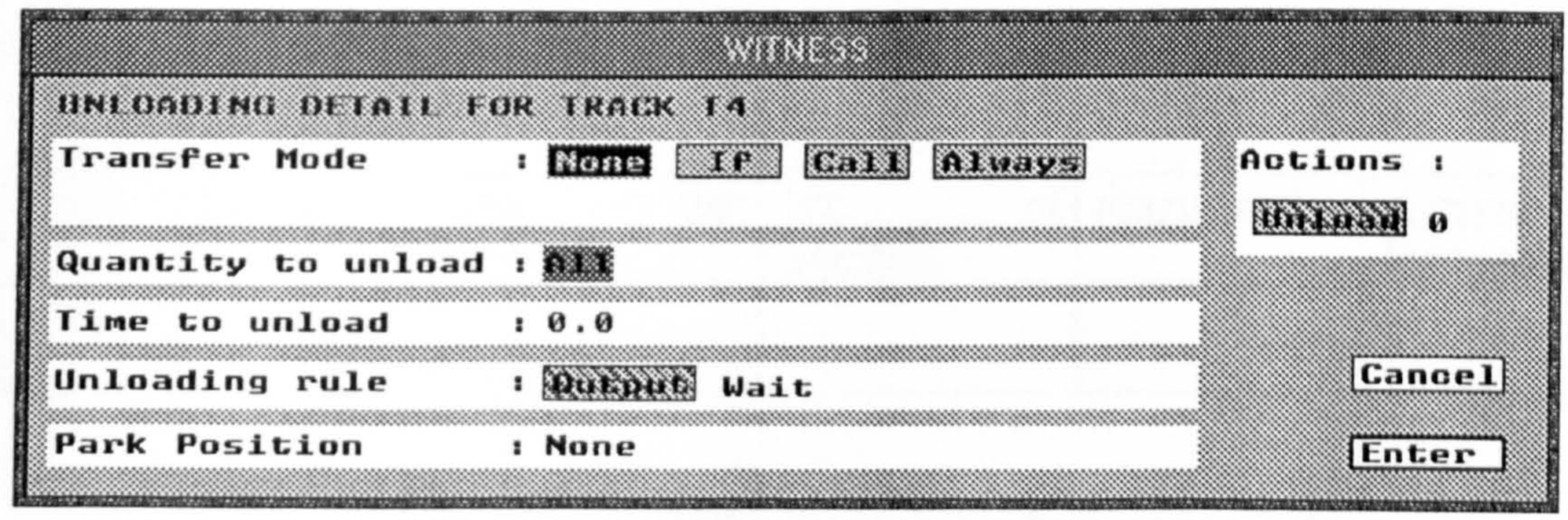


Fig 9 WITNESS Track Unloading Detail Form

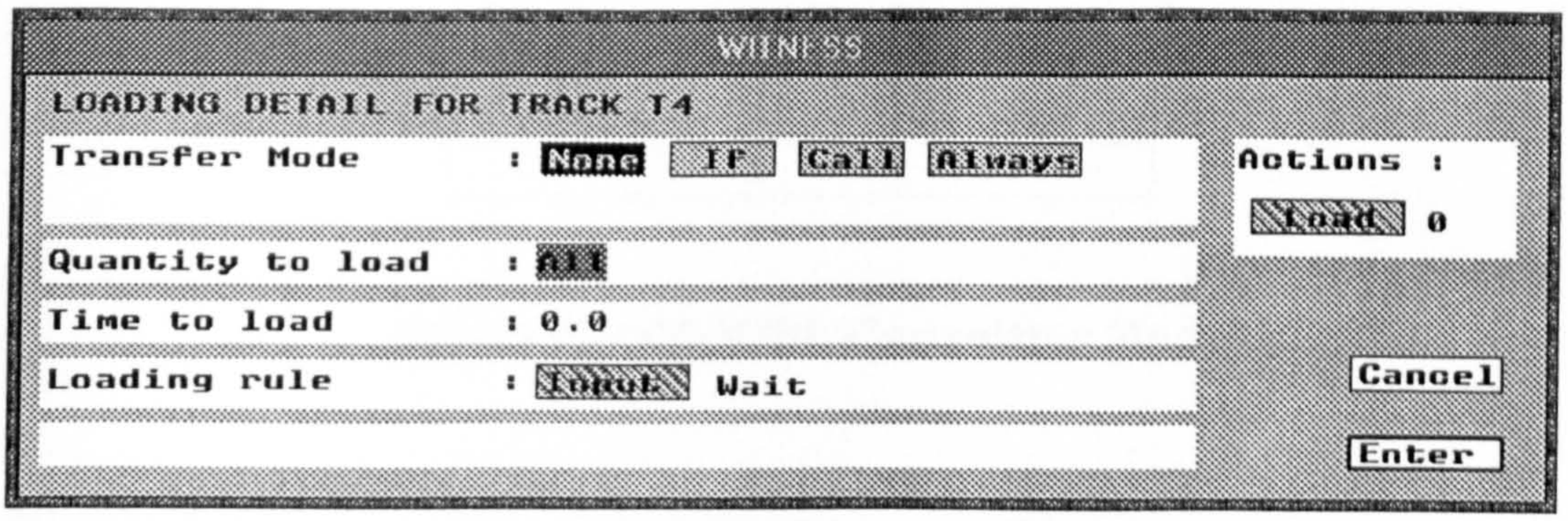


Fig 10 WITNESS Track loading Detail Form

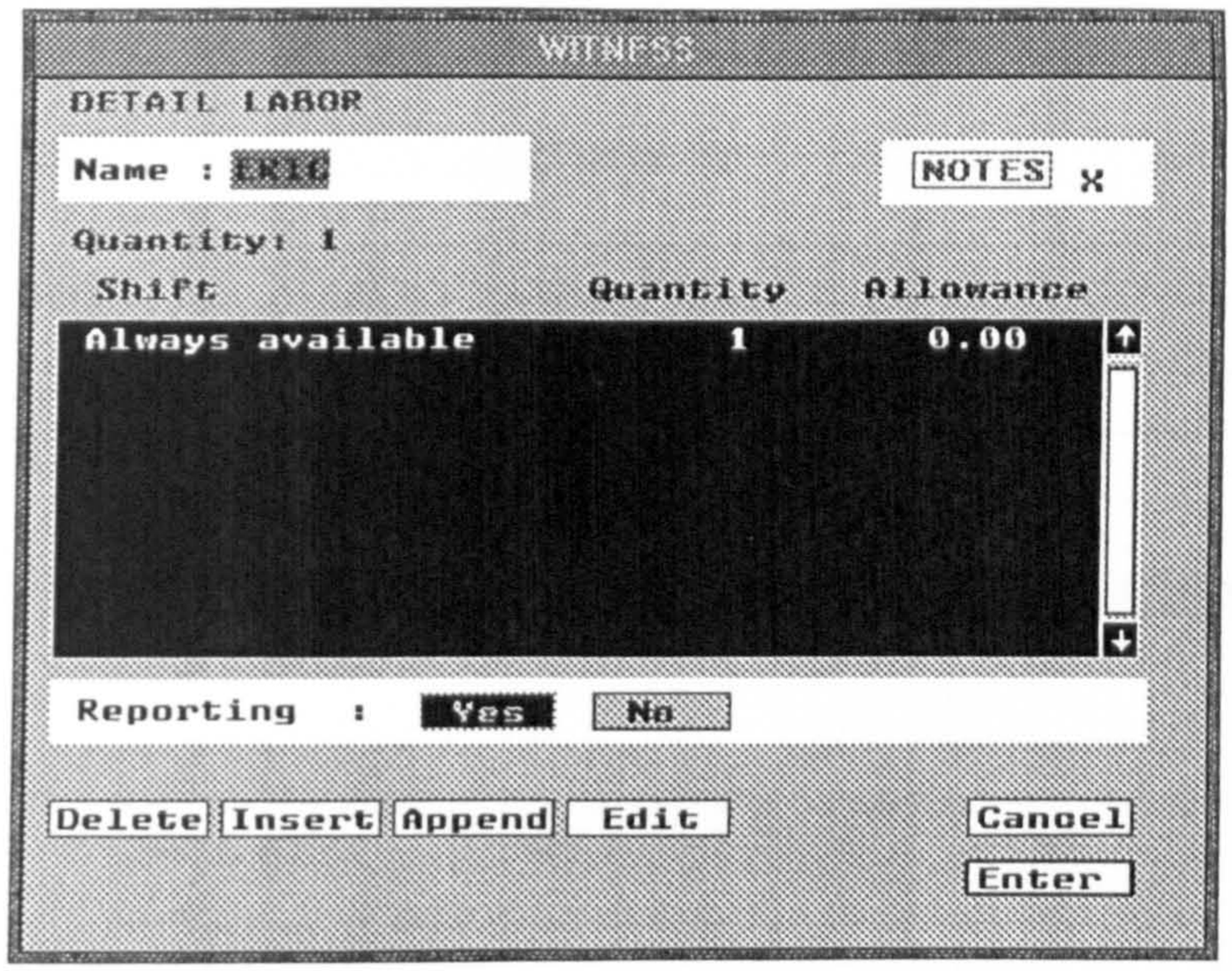


Fig 11 WITNESS Labour Detail Form

Part	Location	Qty per arrival	No of arrivals	start(min)	Arrival frequency(min)
p1					

Fig 12 PROMODEL Part Scheduling Module

Resource	Qty

Fig 13 PROMODEL Capacities Module

Basis	Resource	Part for which setup occurs	Duration (Minutes)	Preceding Part	Qty	Maintenance Resource

Fig 14 PROMODEL Downtimes Module

CONVEYOR SPECIFICATIONS

Section	Type	Speed ft(m)/min	Length of load ft(m)	Spacing ft(m)
con1	T/A			

Fig 15 PROMODEL Conveyor Specifications Sub-module

Location	Conv	Ft(m)	Sec

Fig 16 PROMODEL Conveyor Locations Interface Sub-module

CONVEYOR TRANSFER LOGIC

From	Position ft(m)	To	Position ft(m)	Time (Sec)

Fig 17 PROMODEL Conveyor Transfer Logic Sub-module

TRANSPORTER SPECIFICATIONS FOR(tn)

Speed ft(m)/min	Pickup (sec)	Deposit (sec)	Start position	Search priority	Accel (fpss)	Deccel (fpss)
. . .						

Fig 18 PROMODEL Transporter specification module

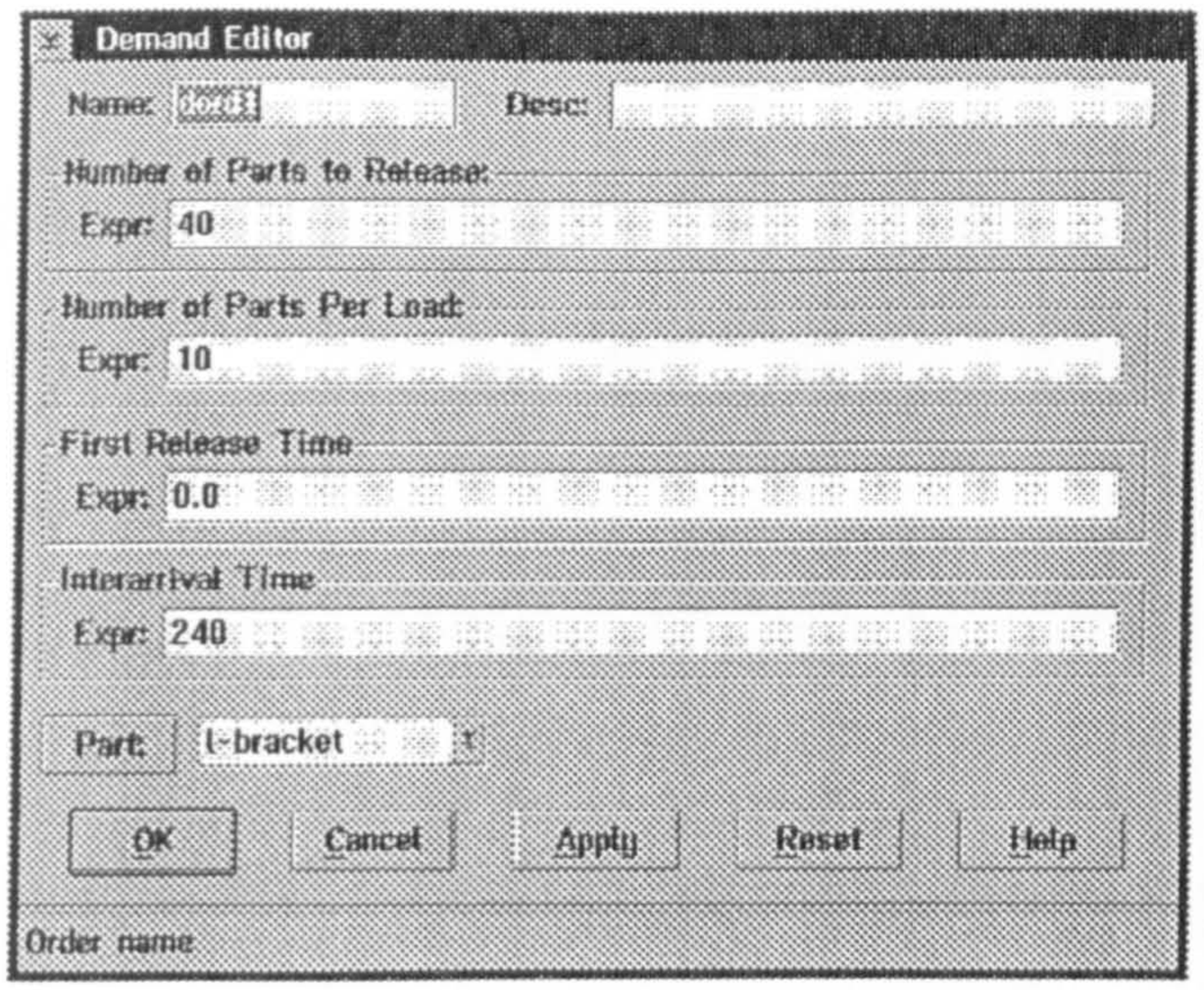


Fig 19 FACTOR/AIM Demand Editor

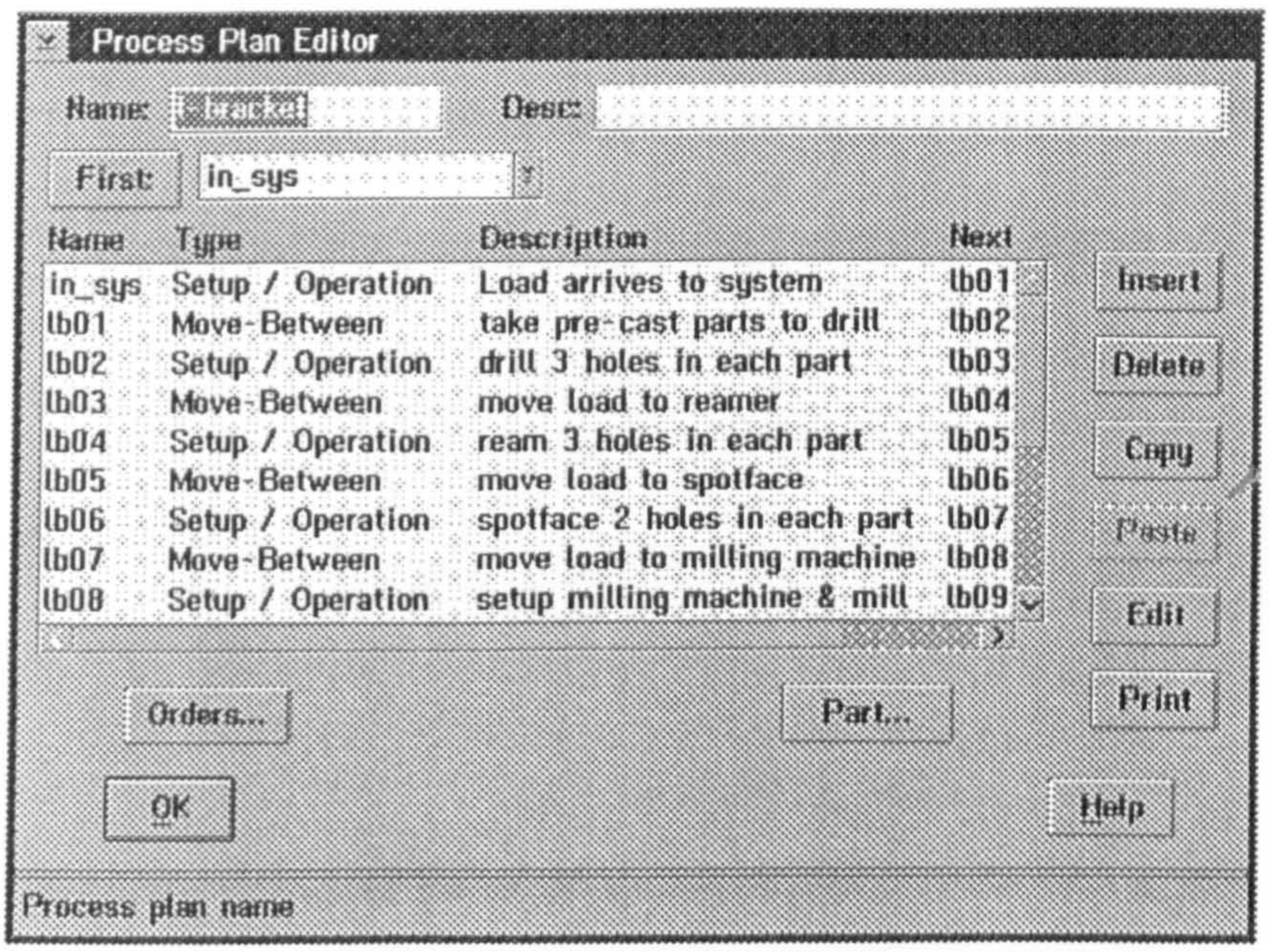


Fig 20 FACTOR/AIM Process Plan Editor

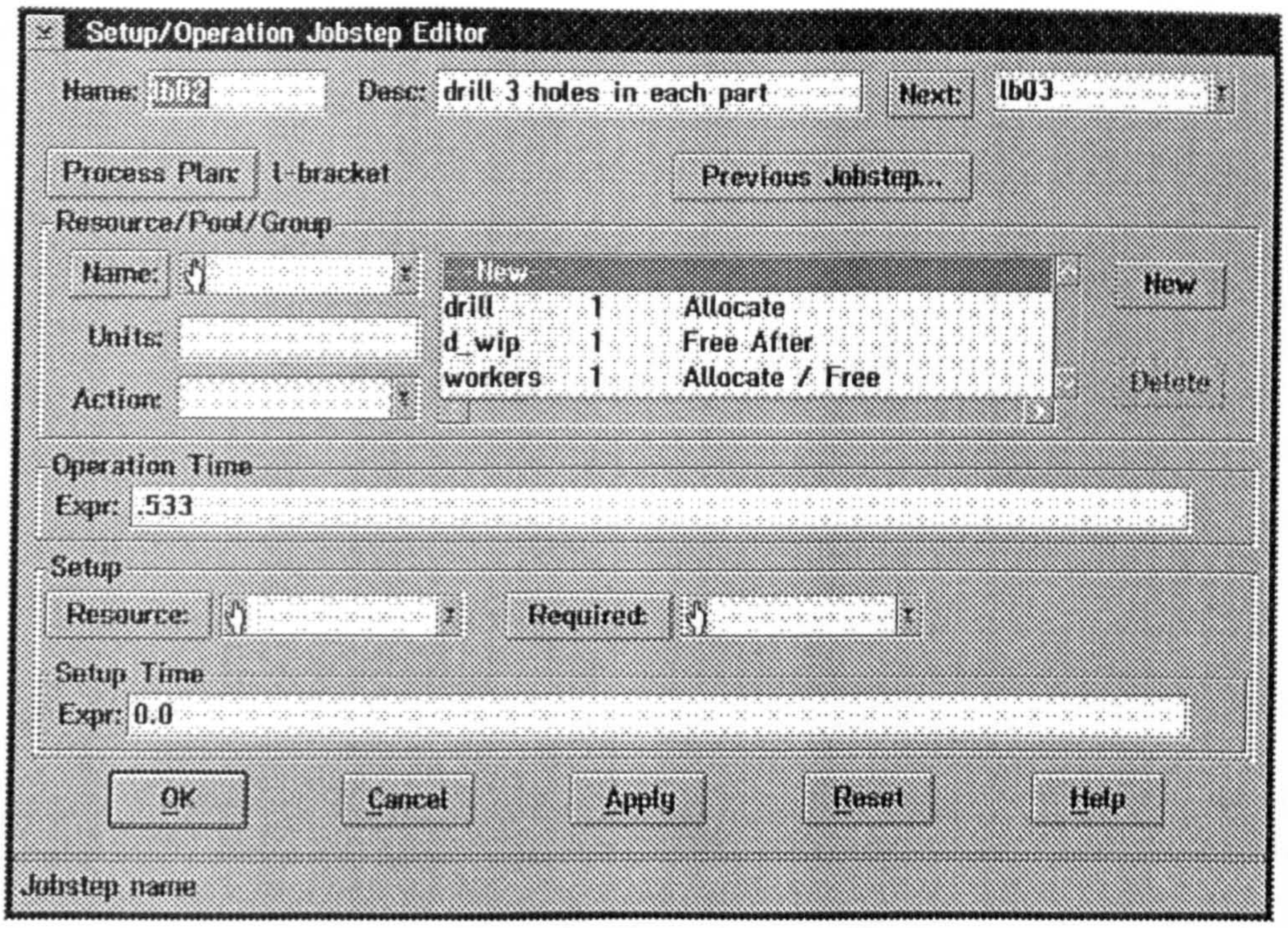


Fig 21 FACTOR/AIM Setup/Operation Jobstep Editor

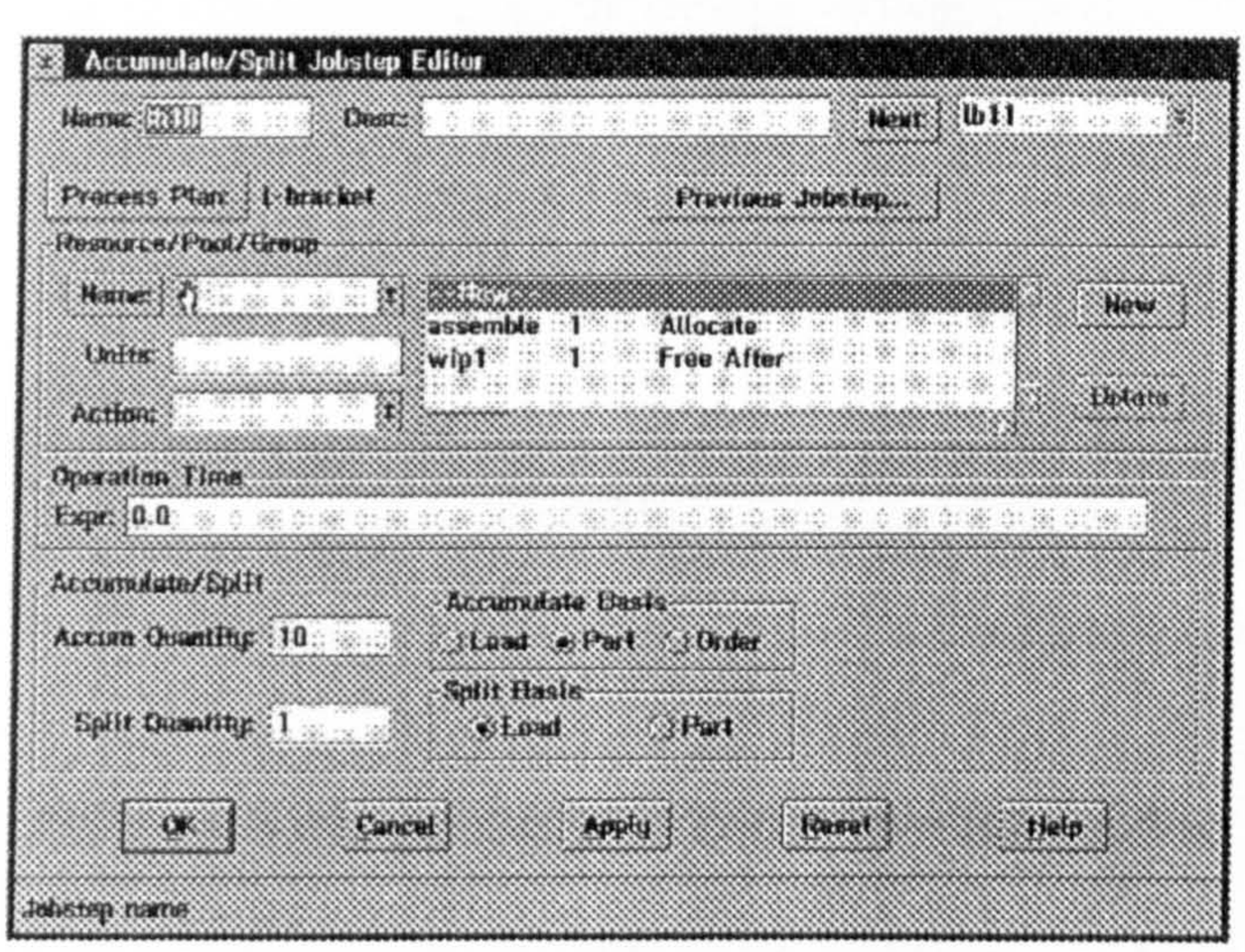


Fig 22 FACTOR/AIM Accumulate/Split Jobstep Editor

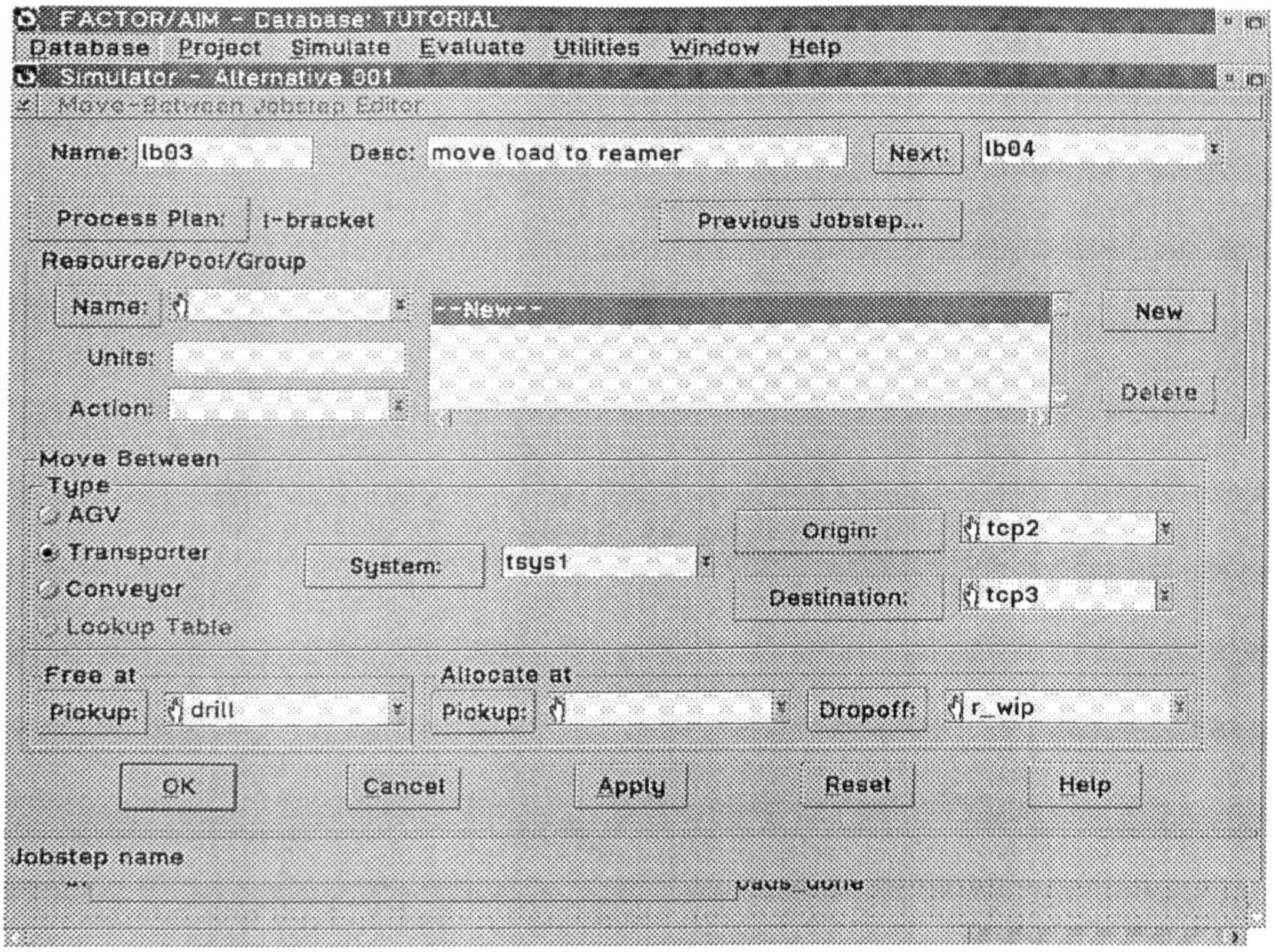


Fig 23 Move-between Jobstep Editor

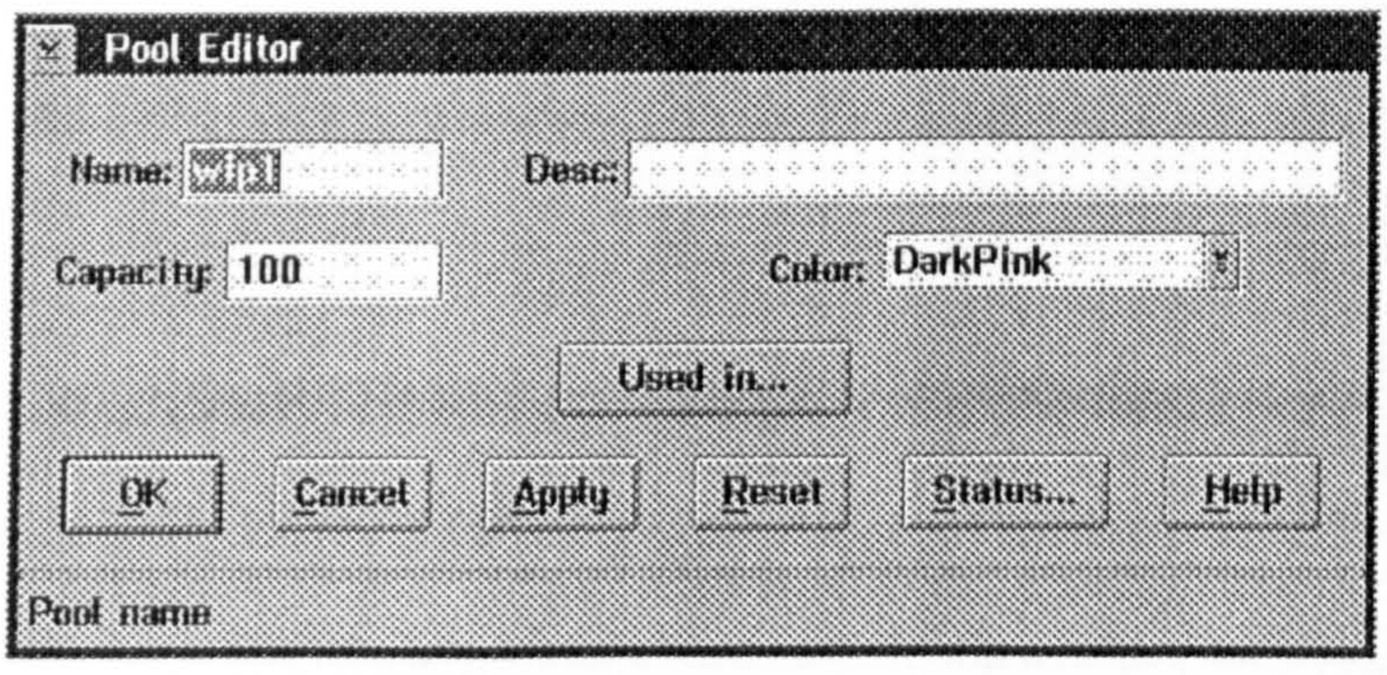


Fig 24 FACTOR/AIM Pool Editor

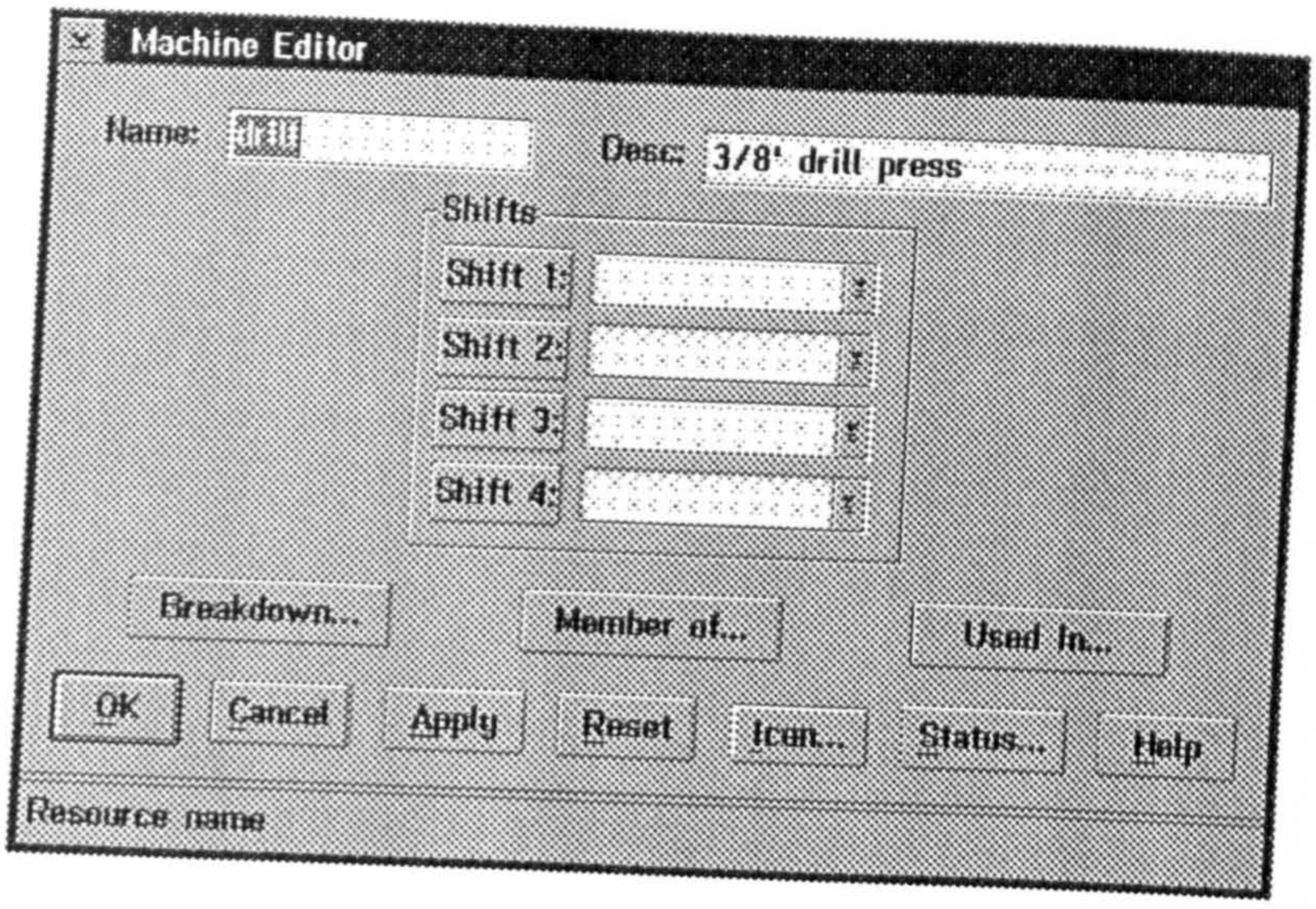


Fig 25 FACTOR/AIM Machine Editor

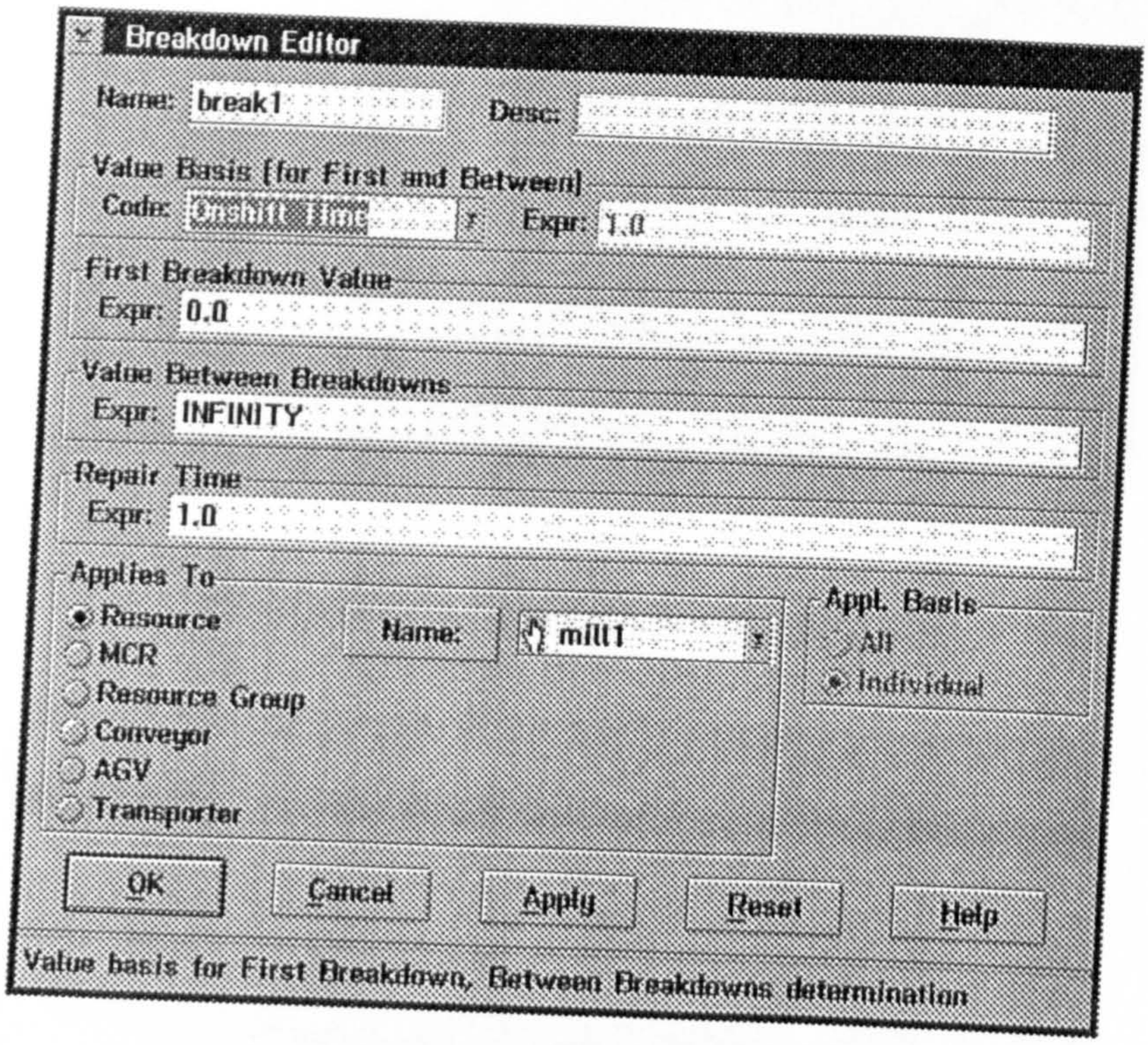


Fig 26 FACTOR/AIM Breakdown Editor

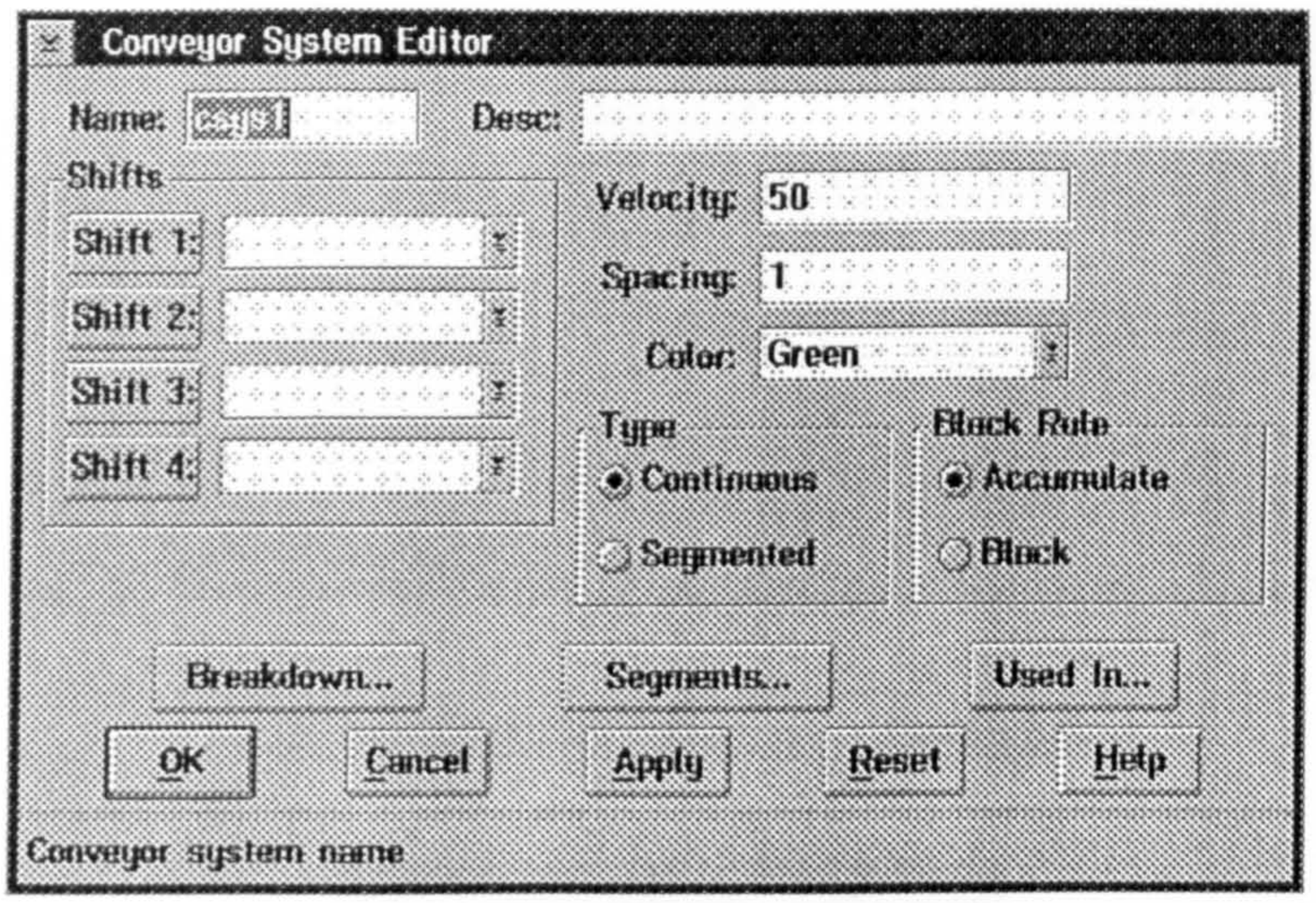


Fig 27 FACTOR/AIM Conveyor System Editor

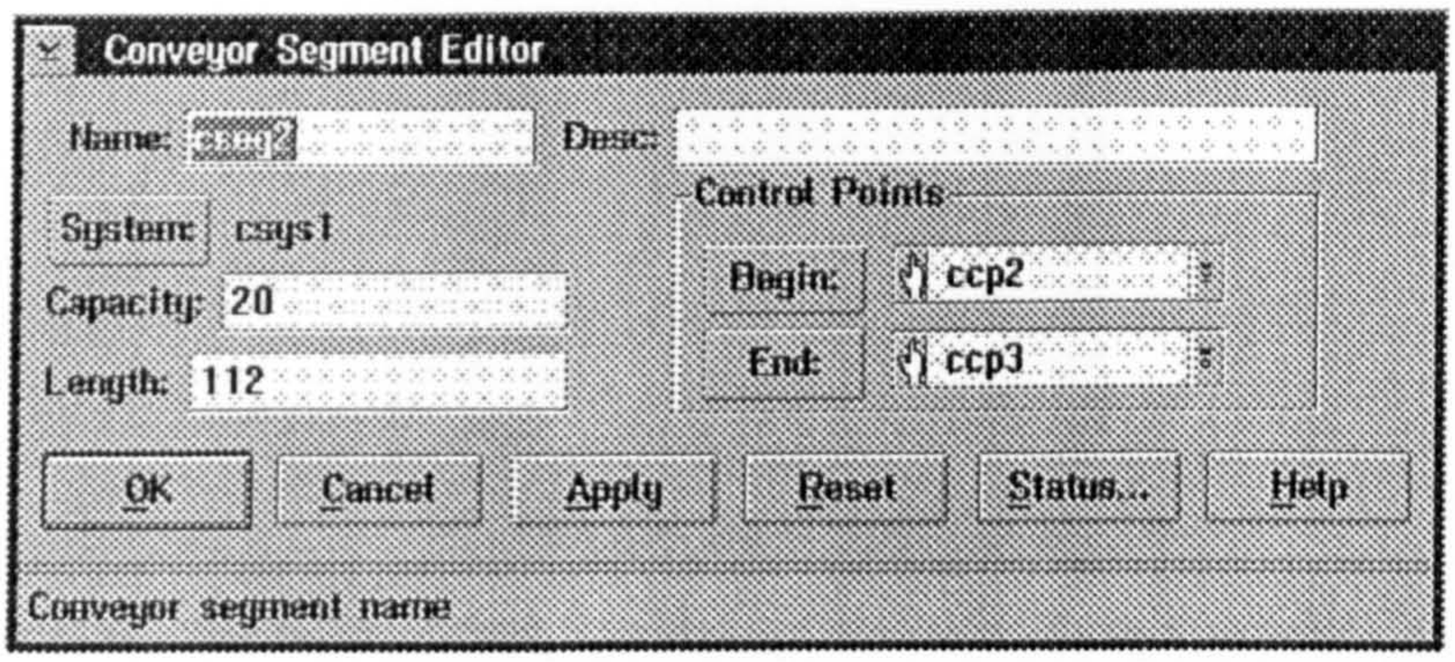


Fig 28 FACTOR/AIM Conveyor Segment Editor

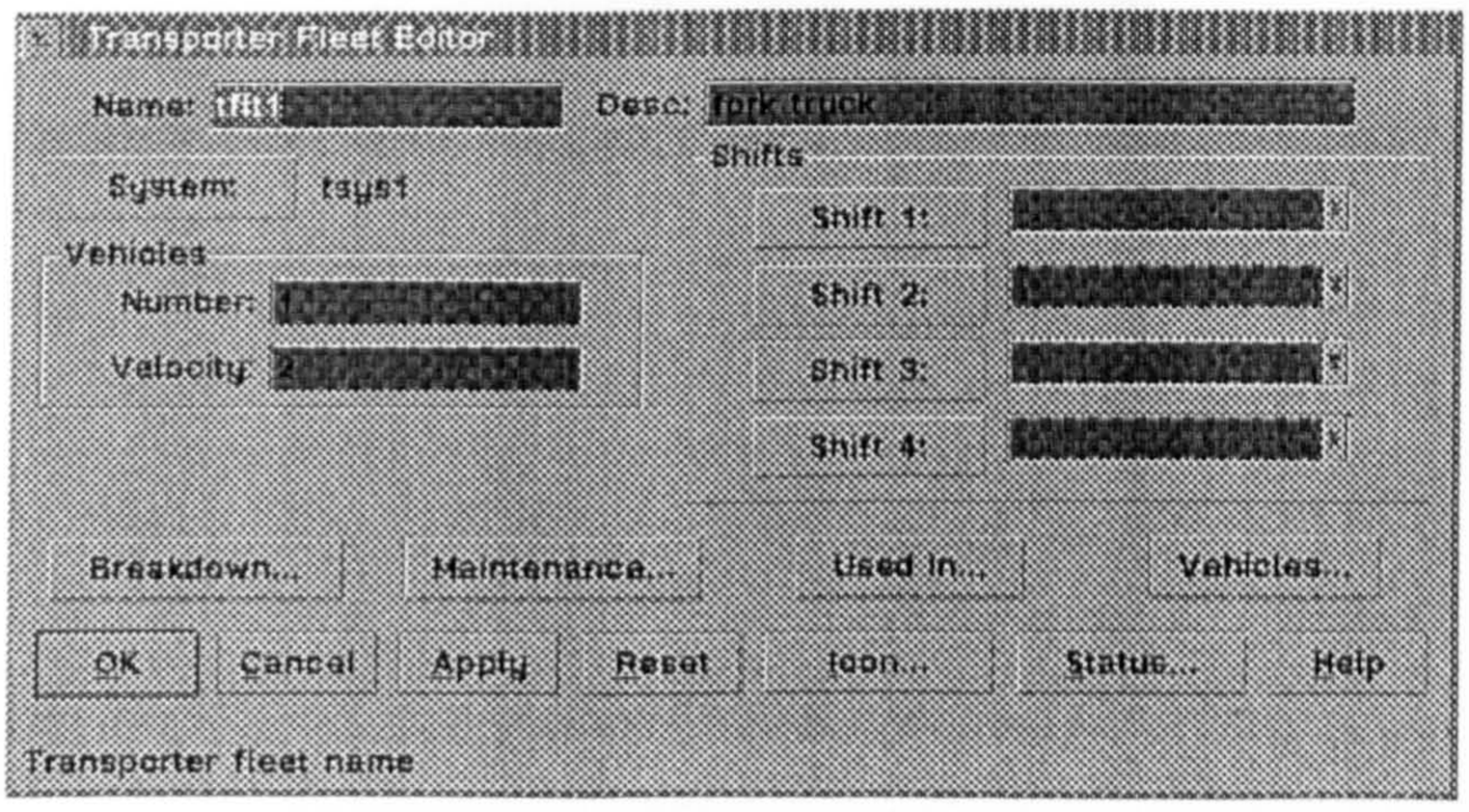


Fig 29 Transporter Fleet Editor

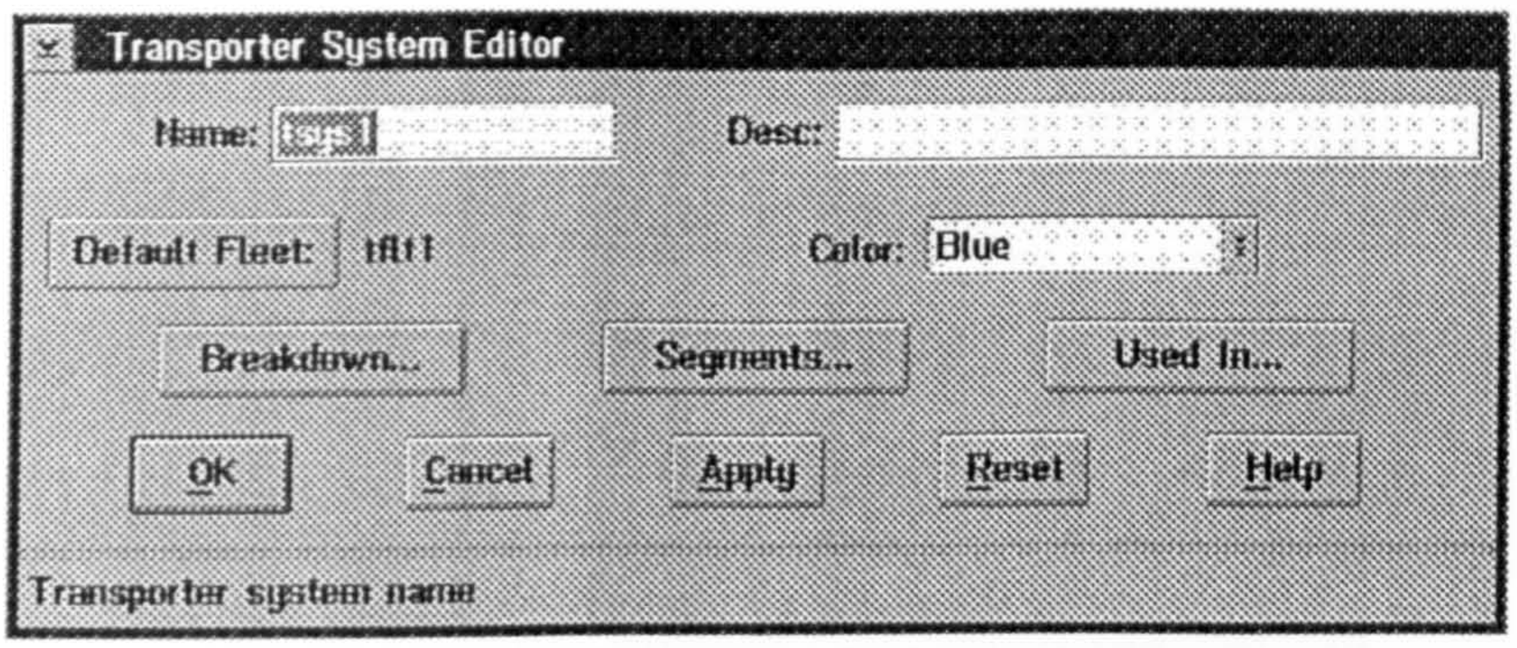


Fig 30 FACTOR/AIM Transporter System Editor

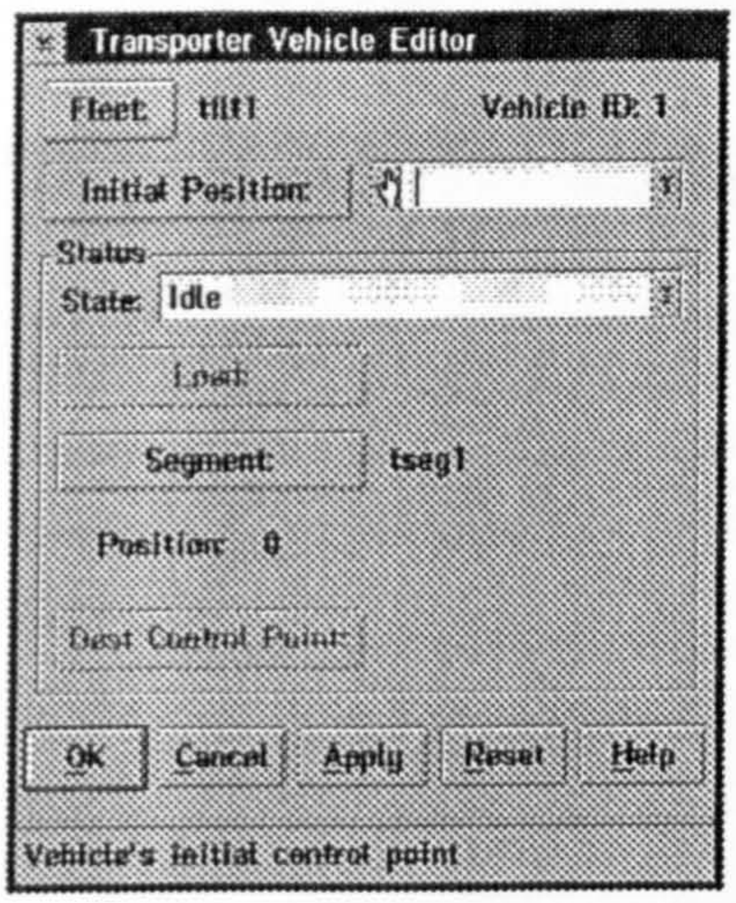


Fig 31 FACTOR/AIM Transporter Vehicle Editor

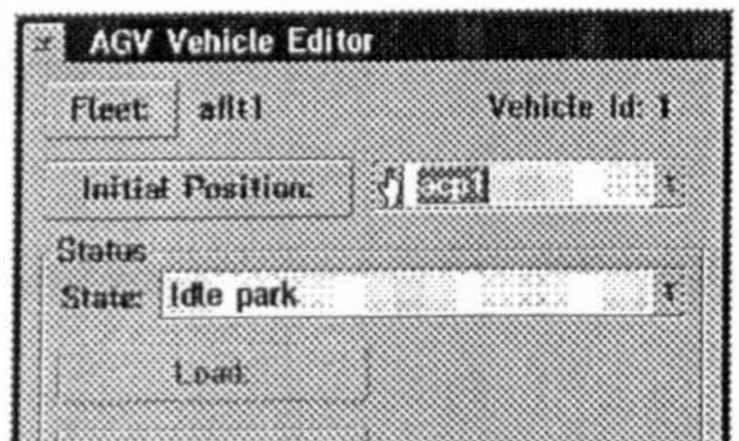


Fig 32 FACTOR/AIM AGV Vehicle Editor

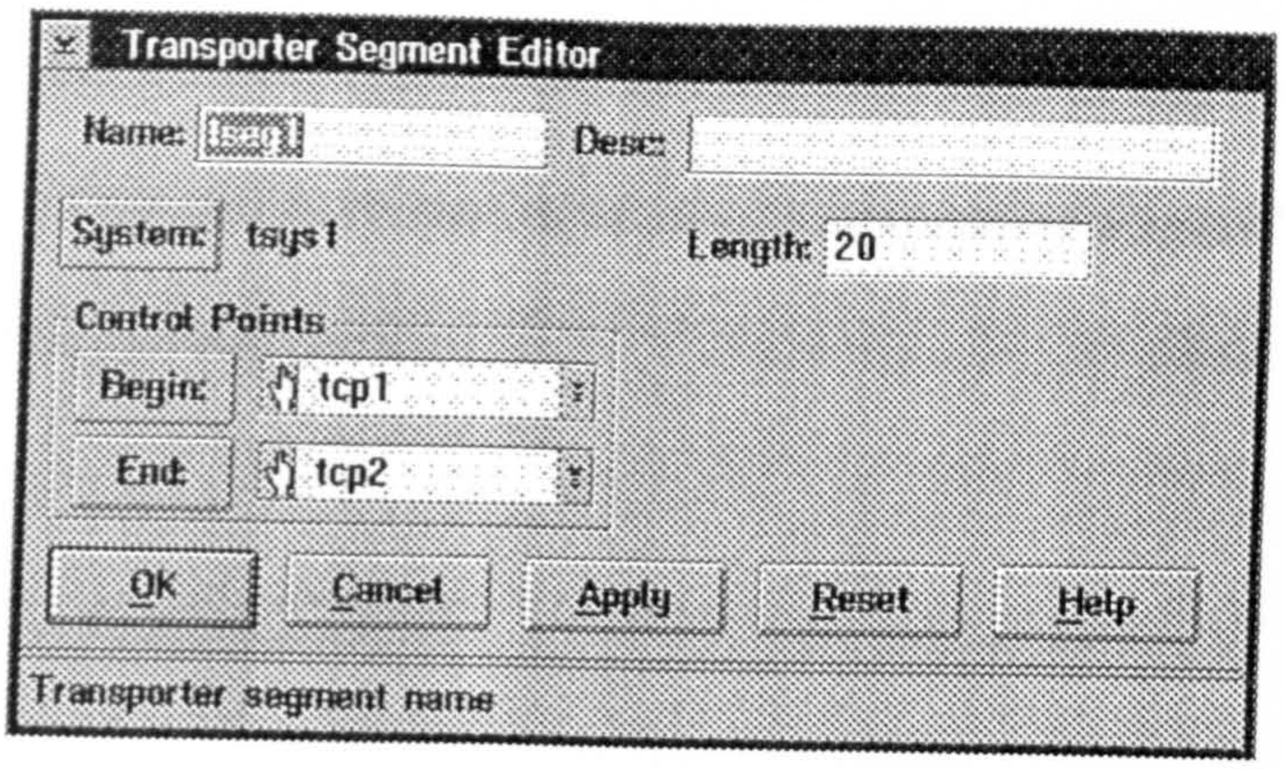


Fig 33 FACTOR/AIM Transporter Segment Editor

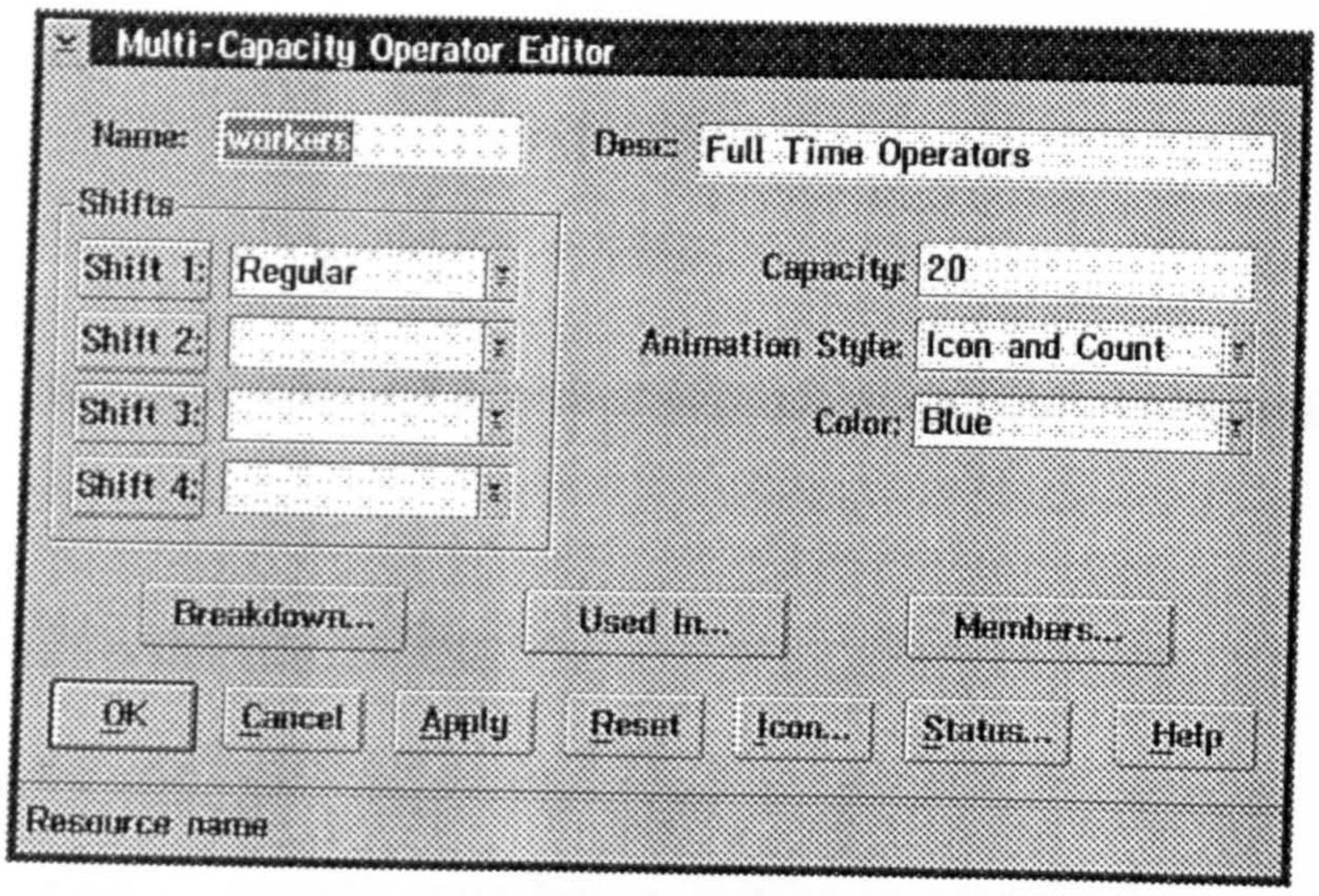


Fig 34 FACTOR/AIM Multi-Capacity Operator Editor

Appendix B. The Remainder of CRMS Frames

This appendix contains the remainder of the frames of the CRMS, other than the one explained in chapter 4.

1 The Buffer Frame of CRMS

The frame for representing the buffer element has the format.

frame(buffer,BUFFER_NO,val,VALUE)

The slots to represent the buffer element's sub-elements are described below.

1.1 The slot for type of buffer

This slot is used to specify whether a buffer is attached to a machine. For example buffer 1 would be specified as a WIP buffer as:

frame(buffer,'1',type,val, 'wip')

A machine attached buffer would be specified as ***input/ouput***. For example buffer 2 could be specified as:

frame(buffer,'2',type,val, 'input/ouput')

1.2 The slot for capacity of buffer

In CRMS the capacity of a buffer is represented in the ***capacity*** slot as:

frame(buffer,'1',capacity,'10')

1.3 The slot for buffer delay

In CRMS the ***buffer*** frame entry for buffer b1, which stores parts for 5 minutes is specified in the ***delay*** slot as:

frame(buffer,'1',delay,'5')

2 Conveyors Frame

This frame has the format:

frame(conveyor,CONVEYOR_NO, val,VALUE)

The slots that are used to represent a conveyor's sub-elements are discussed below.

2.1 The slot for type of conveyor

The type of conveyor '1' would be defined in CRMS in the frame *conveyor* as:

frame(conveyor,'1',type ,val,A or T)

where CONVEYOR_NO=1, SLOT_NAME=A denotes an accumulating conveyor and T a transport conveyor.

2.2 The slot for largest part that can be placed on conveyor

The maximum part length of conveyor '1' is defined in CRMS in the frame *conveyor* as:

frame(conveyor,'1',part_length,val, '15')

where SLOT_NAME=part_length.

2.3 The slot for maximum capacity of conveyor

The maximum capacity (7 in the example below) of conveyor '1' is defined in CRMS in the frame *conveyor* as:

frame(conveyor,'1',max_capacity,val, '7')

where SLOT_NAME=max_capacity.

2.4 The slot for conveyor speed

The speed of conveyor '1' is defined in CRMS in the frame ***conveyor*** as:

frame(conveyor,'1',cycle_time,val, '1')

where SLOT_NAME=cycle_time.

2.5 The slot for conveyer to machine connections

These slots define the connections of conveyors with machines and are found in the CRMS frame ***process_seq***. For example if the processing route of part 1 is represented in CRMS as:

frame(process_seq,'1',0,load / unload,val,'1')
frame(process_seq,'1',1,machine_no,val,'1')
frame(process_seq,'1',1,conveyor_no,val,'1')
frame(process_seq,'1',1,proc_time,val,'3')
frame(process_seq,'1',1,setup_time,val,'4')
frame(process_seq,'1',2,machine_no,val,'2')
frame(process_seq,'1',2,conveyor_no,val, '2')
frame(process_seq,'1',2,proc_time,val,'3')
frame(process_seq,'1',2,setup_time,val,'4')
frame(process_seq,'1',3,machine_no,val,'3')
frame(process_seq,'1',3,conveyor_no,val,'2')
frame(process_seq,'1',3,proc_time,val,'1')
frame(process_seq,'1',3,setup_time,val,'4')
frame(process_seq,'1',4,load/unload,val,'1')

frame(process_seq,'1',1,conveyor_no,val,'1')

it means that machine m1 is connected to conveyor c1, machine m2 to conveyor c2, machine m3 to conveyor c3 and the load/unload station to conveyor c1.

3. Vehicles Frame

This frame has the format:

frame(Vehicle,VEHICLE_NO, val,VALUE)

The slots that are used to represent a vehicle's sub-elements are discussed below.

3.1 The slot for maximum number a vehicle can carry

In CRMS the maximum number (3 in example below) a vehicle (1 in the example) can carry is represented in the frame ***vehicle***, using the ***capacity*** slot as:

frame(vehicle,'1',capacity,val,3)

Where VEHICLE_NO=1 and SLOT_NAME=capacity.

3.2 The slots for vehicle speed when it is loaded and unloaded.

This is represented in the frame ***vehicle***, via the ***load_speed*** and ***unload_speed*** slots as:

```
frame(vehicle,'2',load_speed,val,10)
frame(vehicle,'2',unload_speed,val,7)
```

Here VEHICLE_NO=2 has SLOT_NAME=load_speed and unload_speed to represent its loaded (10 in example) and unloaded speeds (7 in example).

3.3 The slot fors vehicle acceleration and deceleration

The acceleration and deceleration of a vehicle is represented in the frame ***vehicle***, via the accel and decel slots as:

```
frame(vehicle,'1',accel,val, '10')
frame(vehicle,'1',deccel,val, '7')
```

Here VEHICLE_NO=2 has SLOT_NAME=acc and decel to represent the acceleration (10 in example) and deceleration (7 in example).

3.4 The slot for start position of a transporter

This is represented in CRMS in the frame ***vehicle***, via the ***start*** slot as:

frame(vehicle,'1',start,val, '10')

Here VEHICLE_NO=1 has SLOT_NAME=start to represent its start position (track 10 in example).

3.5 The slot for pickup time

This is represented in CRMS in the frame vehicle, via the ***pickup*** slot as:

frame(vehicle,'1',pickup,val, '10')

Here VEHICLE_NO=1 has SLOT_NAME=pickup to represent its pickup time (10 in example).

3.6 The slot for deposit time

This is represented in CRMS in the frame ***vehicle***, via the ***deposit*** slot as:

frame(vehicle,'1',deposit,val, '10')

Here VEHICLE_NO=1 has SLOT_NAME=deposit to represent its deposit time (10 in example).

4. Track Frame

This frame has the format:

frame(track,TRACK_NO, val,VALUE)

The slots that are used to represent a track's sub-elements are discussed below.

4.1 The slot for track to track connection

The ***connect_to*** slot identifies which tracks are connected to which. For example if we wished to specify that track 1 was connected to track 2, in CRMS this would be represented in the frame ***vehicle*** as:

frame(track,'1',connect_to,val, '2')

Here TRACK_NO=1 has SLOT_NAME=connect_to signify track 1 is connected to track 2.

4.2 The slot for length of the track

In CRMS the length of two connecting tracks tr1 and tr2 would be defined in the frame ***track*** via the ***length*** slot as:

```
frame(track,'1',length,val, '10')
frame(track,'2',length,val, '7')
```

Here TRACK_NO=1 has SLOT_NAME=length (10 in example), and TRACK_NO=2 has SLOT_NAME=length (7 in example).

4.3 The slot for maximum speed

In CRMS the maximum speed a vehicle can travel on a track is represented in the frame ***track*** as:

```
frame(track,'1',max_speed,val, '10')
```

Here TRACK_NO=1 has SLOT_NAME=max_speed to represent its maximum speed (10 in example).

4.4 The slot for track to machine connections

In CRMS the connection of tracks to machine would be specified in the ***process_seq*** frame using the ***load_track*** and ***unload_track*** slots. For example

```
frame(process_seq,'1',1,machine_no,val,'3')
frame(process_seq,'1',1,load_track ,val,'1')
frame(process_seq,'1',1,unload_track ,val,'3')
frame(process_seq,'1',2,machine_no,val,'4')
frame(process_seq,'1',2,load_track ,val,'3')
frame(process_seq,'1',2,unload_track ,val,'5')
```

signifies that track 3 is connected to machine 3 and track 5 is connected to machine 4.

where

- PART_NO=1
- VISIT_NO=1 and 2
- SLOT_NAME=machine_no, load_track , unload_track , machine_no,

  load_track or unload_track in turn.

5 Manpower Frame

This frame has the format:

```
frame(manpower,TRACK_NO, val,VALUE)
```

The slots that are used to represent the sub-elements of labour are discussed below.

5.1 The slot for manpower quantity

The quantity of a certain type of manpower would in CRMS be represented as an entry in ***quantity*** slot:

frame(manpower,quantity,val,'5')

where SLOT_NAME=quantity (5 in example).

5.2 The slot for manpower assignment

If man number '1' is required for machine '1' to operate this would be defined in CRMS in the frame ***machine*** as:

frame(machine,'1',lab_for_cyc,val,'1')

Here MACHINE_NO= 1 and SLOT_NAME=lab_for_cyc.

The manpower required for machine '1' to setup would be defined in CRMS in the frame ***machine*** as:

frame(machine,'1',lab_for_setup,val,'1')

Here MACHINE_NO= 1 has SLOT_NAME=lab_for_setup for labour required to setup (man 1 in above example).

Appendix C Model Construction Rules

This appendix comprises WITNESS, PROMODEL and FACTOR/AIM part and machine modelling rules and their explanation. They provide additional explanation of the operation of the translation system.

1 Witness model contruction rules

1.1. WITNESS Part modelling rules

The first rule for modelling a part, as an example, is used to define the part. The rule has the form:

```
rule 1#:
    [1:frame(creation_info,A,lot_size,val,B)]
    =>
    [retract(all),          %retracts matching frame to avoid continuous execution
    tell('model.lst'),          %opens file model.lst for output
    write('PART: p'),           %write p to denote part
    write(A),                   %writes  part number
    write(',Variable attributes;'),
    nl].                        %writes newline
```

The portion of CRMS that triggers it is:

```
frame(creation_info,1,lot_size,val,10)
```

which would generate a part definition:

PART: p1,Variable attributes;

The rule for transferring the data characteristics of parts into a WITNESS list file is:

```
rule 1#:

[frame(creation_info,A,lot_size,val,B),
frame(creation_info,A,first_arr_tim,val,C),
frame(creation_info,A,max_arr,val,D),
frame(creation_info,A,inter_arr_tim,val,E),
frame(creation_info,A,no_stream,val,F)]
=>
[retract(all),     %retracts all data that matches LHS to prevent continuous execution of rule
tell('model.lst'),                    %setups up file model.lst as current output stream.
write('p'),                           %writes character p to denote part to file.
write(A),                             %writes part number
nl,                                   %writes newline
nl,                                   %writes newline
write('NAME OF PART: p'),
write(A),
write(';')nl,
write('TYPE: Variable attributes;')nl,
write('GROUP NUMBER: 1;')nl,
write('MAXIMUM ARRIVALS: '),
write(D),
write(';')nl,
write('INTER ARRIVAL TIME: '),
write_dist(A,E,F),                    %user written action for testing distribution type and
write(';')nl,                           writing it and its parameters.
write('FIRST ARRIVAL AT: '),
write(C),
write(';')nl,
write('LOT SIZE: '),
write(B),
write(';')nl,
write('OUTPUT RULE:PUSH to ROUTE;')nl,
route(A),          %User written predicate for generating part route in part detail
nl,
write('END')nl,
write('REPORTING: Yes;')nl,
write('CONTAINS FLUIDS: No;')nl,
write('SHIFT: Undefined;')nl,nl,
write('END p'),
write(A),nl,nl].
```

The left hand side of this rule is matched against CRMS data of the form shown below, which in this case is the data characteristics of part 1:

```
fact(frame(creation_info,'1',lot_size,val,'1')).
fact(frame(creation_info,'1',first_arr_tim,val,'1')).
fact(frame(creation_info,'1',max_arr,val,'2')).
fact(frame(creation_info,'1',inter_arr_tim,val,integer_uniform_distribution')).
fact(frame(creation_info,'1',p1,val, '1')).
fact(frame(creation_info,'1',p2, '10')).
fact(frame(creation_info,'1',p3 , '1')).
```

The rule includes a number of right hand side actions in the form of Prolog predicates, which manipulate and transfer data from CRMS, via the Prolog clause ***store***, into the list file. These user defined actions have the advantage of limiting the number of rules required for transformation, and only a single rule was required for transferring part data.

For the above part detail rule only two user written predicates are needed **write_dist_3 and route_1**:

- **write_dist_3** is used to test and write a probability distributions and its characteristics. It has 15 different clauses, one for each distribution The one for the IUNIFORM distribution tests to see if E equals integer_uniform_diistribution (tests to see if the inter-arrival time distribution is an integer uniform) and, if it is, it proceeds to extract from the Prolog clause ***store*** its parameters and asserts them into the inter-arrival time statement of the part detail in the list file.

```
write_dist(A,E,F):-E=integer_uniform_distribution>%test for distribution
fact(frame(creation_info,A,p1,val,X)),        %access 1st distribution parameter
fact(frame(creation_info,A,p2,val,Y)),        %access 2nd distribution parameter
```

```
fact(frame(creation_info,A,p3,val,Z)),          %access 3rd distribution parameter
write('IUNIFORM('),                                %writes IUNIFORM
write(X),                                          %write 1st  parameter
write(','),
write(Y),                                          %write 2nd parameter
write(','),
write(Z),                                          %write 3rd parameter
write(');').
```

The above action tests to see if E= integer_uniform_distribution is true. If it is then X and Y are instantiated to the part's inter-arrival time parameters p1 and p2. It then proceeds to write **IUNIFORM** followed by the parameters p1 and p2 as part of the inter--arrival time statement of part 1.

- **route_1** given below generates, in the list file, the processing sequence of a part from data in CRMS of the form:

```
fact(frame(entry_point,'1',_,load / unload,val,'1')).
fact(frame(process_seq,'1',1,machine_no,val,'1')).
fact(frame(process_seq,'1',1,conveyor_no,val,['1'])).
fact(frame(process_seq,'1',1,proc_time,val,'3')).
fact(frame(process_seq,'1',1,setup_time,val,'4')).
fact(frame(process_seq,'1',2,machine_no,val,'2')).
fact(frame(process_seq,'1',2,conveyor_no,val,['2'])).
fact(frame(process_seq,'1',2,proc_time,val,'3')).
fact(frame(process_seq,'1',2,setup_time,val,'4')).
fact(frame(process_seq,'1',3,machine_no,val,'3')).
fact(frame(process_seq,'1',3,conveyor_no,val,['3'])).
fact(frame(process_seq,'1',3,proc_time,val,'3')).
fact(frame(process_seq,'1',3,setup_time,val,'4')).
fact(frame(exit_point,'1',_,load / unload,val,'2')).
fact(frame(process_seq,'1',4,conveyor_no,val,['4'])).
```

The right hand side action that transfers this information into the part route section of a part details in the WITNESS list file:

```
route(A):-
fact(frame(entry_point,A,_,load / unload,val,B)),  % access frames for entry point of
                                                      part
retract(fact(frame(entry_point,A,_,load / unload,val,B))),
write('PART ROUTE:STAGE 1:load'),                  %sets stage 1 as load
write(B),
write(';'),
retractall(stage(X)),                              % access stage in part route
assert(stage(1)),
(test_rout_end(A);                                 % predicate that tests if end of route
true),
exit_point(A).                                     % predicate that puts exit point of part in route
```

It includes user written predicates **test_rout_end(A)** and **exit_point(A),** where **test_rout_end_1** retrieves the machines visited and conveyors used to get the part to the machines. Since the part may get to a machine via a number of conveyors, the **test_end_con_1** predicate is used to include all the conveyors in the route up to the last one which delivers the part to the machine.

```
test_rout_end(A):-
retract(fact(frame(process_seq,A,B,machine_no,val,C))),  % aceess frames that contain
                                                           the    machine visit number,
retract(fact(frame(process_seq,A,B,conveyor_no,val,D))),     conveyor number,
retract(fact(frame(process_seq,A,B,proc_time,val,E))),       processing time and
retract(fact(frame(process_seq,A,B,setup_time,val,F))),      setup time
inc_stage,                                                 % stage number
stage(W),                                                  % retrieve stage number W
nl,
write('      STAGE '),
write(W),                                          % write stage number retrieved W
write(':m'),
write(C),write(';'),nl,
write('                R_SETUP:'),
write(F),write(';'),nl,                                    %write setup time E
write('                R_CYCLE:'),
write(E),write(';'),                                       %write cycle time F
test_rout_end(A).                          % redo and if it carnt retrive any frames
                                            for part A control  is returned to predicate route above
```

The predicate **exit_point_1** adds the unload station and the conveyor(s) for transporting the part there.

```
exit_point(A):-
fact(frame(exit_point,A,_,load/unload,val,C)), % retrieve exit point of part A
fact(frame(process_seq,A,B,conveyor_no,val,D)),    % retrieve conveyor(s) for getting
                                                    there
inc_stage,                                          % increment stage
nl,
write('     STAGE '),
stage(W),                                           % retrieve current stage
write(W),                                           % write stage W
write(':load'),
write(C),
write(';'),nl,
write('              R_SETUP:0;'),nl,
write('              R_CYCLE:0;'),
stage,
nl,
write('     STAGE '),
stage(X),                                   % access SHIPPING stage
write(X),                                   % write SHIIPPING stage
write(':SHIP;'),nl,
write('              R_SETUP:0;'),nl,
write('              R_CYCLE:0;').
```

This transformation of CRMS data, shown above would create a part detail like:

```
p1

Name of part: p1;
Type: Variable attributes;
Group number: 1;
Maximum arrivals: 2;
Inter arrival time: IUNIFORM(1,10,1);
First arrival at: 1.0;
Lot size: 1;
Output rule: PUSH to ROUTE;
Part route: STAGE 1 : load/unload1;
                      R_SETUP : 0;
                      R_CYCLE : 0;
            STAGE 2 : conveyor1@0;
                      R_SETUP : 0;
                      R_CYCLE : 0;
```

```
STAGE 3 : m1;
          R_SETUP : 4;
          R_CYCLE : 3;
STAGE 4 : conveyor2@0;
          R_SETUP : 0;
          R_CYCLE : 0;
STAGE 5 : m2;
          R_SETUP : 4;
          R_CYCLE : 3;
STAGE 6 : conveyor3@0;
          R_SETUP : 0;
          R_CYCLE : 0;
STAGE 7 : m4;
          R_SETUP : 4;
          R_CYCLE : 3;
STAGE 8 : conveyor4@0;
          R_SETUP : 0;
          R_CYCLE : ;
STAGE 9 : load/unload2;
          R_SETUP : 0;
          R_CYCLE : 0;
End
Reporting: Yes;
Contains fluids: No;
SHift: Undefined;

END p1
```

1.2 WITNESS Machine Modelling Rules

The application of rule 1 below generates the machine definition.

```
rule 1#:
[frame(machine_info,A,type,val,B),
frame(machine_info,A,quantity,val,C),
frame(machine_info,A,in_bufffer,val,D)]
=>
[retract(all),
tell('model.lst'),
write('MACHINE:m'),          %write MACHINE:m to denote machine
write(A),                    %write machine number
write(','),
write(C),                    %write quantity of particular machine
write(','),
write(B),          %write machine whether single, assembly or production
write(','),
```

```
write(D),                          %write input buffer size
write(',0,1;'),
nl].
```

The application rule 2 below generates the machine detail.

```
rule 2#:
[frame(machine_info,A,type,val,T),
frame(machine_info,A,quantity,val,B),
frame(machine_info,A,lab_priority,val,C),
frame(machine_info,A,lab_for_rep,val,E),
frame(machine_info,A,lab_for_cyc,val,F),
frame(machine_info,A,breakdown_type,val,M),
frame(machine_info,A,setup_type,val,H),
frame(machine_info,A,in_bufffer,val,I)]
=>
[retract(all),
tell('model.lst'),
write('m'),                                %write m to denote machine
write(A),                                  %write machine number
nl,nl,
write('NAME OFMACHINE:m'),                 %write machine title
write(A),                                  %write machine number
write(';')nl,
write('QUANTITY:'),
write(B),                                  %write machine quantity
write(';')nl,
write('TYPE:'),
mach_type(A,T),                %action for writing machine type andthier charateristics
nl,
write('PRIORITY:undefined
write('LABOR:'),nl,
write('    Repair: man'),                  %write labour man number
write(E),
write(';'),nl,
write('    Pre-empt level: None;'),nl,
write('END'),nl,
write('LABOR:'),nl,
write('    Cycle: man'),write(F),write(';')nl,
write('    Pre-empt level: None;'),nl,
write('END'),nl,
write('DISCRETE LINKS:'),nl,
write('    Fill:None'),nl,
write('END'),nl,
write('DISCRETE LINKS:'),nl,
write('    Empty:None'),nl,
write('END'),nl,
```

```
setup(A,H),nl,                                    %action for writing setup info
write('CYCLE TIME:R_CYCLE;'),nl,
breakdown(A,M),nl,                                %action for writing breakdown info
write('INPUT RULE: BUFFER ('),write(I),write(');'),nl,
write('OUTPUT RULE: PUSH to ROUTE;'),nl,
write('REPORTING: Individual;'),nl,
write('SHIFT: Undefined,0,0;'),nl,nl,
write('END m'),write(A),nl,nl].
```

The rule contains user written actions or predicates **machine_type_2, Priority _2, setup_2 and breakdown_2**. Where:

- There are three clauses for the **setup_2**. The one for a setup after a part change is :

```
setup(A,H):-H='2 After a part change'->     %checks to see if setup after part change
fact(frame(machine_info,A,lab_req,val,B)),  %retrieve frame with labour requirement
write('SETUP_DETAIL'),nl,
write('  Setup number:1'),nl,
write('  *Mode:Part change;'),nl,
write('  *Setup time:R_SETUP;'),nl,
write('  *Description:Setup number 1'),nl,
write('  *Station number:1;'),nl,
write('  LABOR:'),nl,
write('   Setup: man'),
lab_tip(B),                                       %action for writing labour setup
write(';'),nl,
write('  END'),nl,
write('END SETUP_DETAIL'),nl,
retract(fact(frame(machine_info,A,lab_req,val,B))).
```

the others, near identical in form, are for when there is **no setup** and when there is a **setup after a number of operations**.

- Their are four clauses for the **breakdown_2** predicate, where the one for a breakdown after a number of operations is :

```
breakdown(A,G):-G='4 After a number of operations'->%check to see breakdown after
write('BREAKDOWNS: Operations;'),nl,                 number ops
fact(frame(machine_info,A,nof_operations,val,B)),   %retrieve frames with breakdown
```

```
fact(frame(machine_info,A,rep_time,val,C)),                    information
fact(frame(machine_info,A,scrap_part,val,D)),
write('*Ops between breakdown:'),
write(B),write(';')nl,                                  %write number of operations
write('* Repair time:'),
write(C),write(';')nl,                                  %repair time
write('* Scrap part:'),write(D),write(';')nl,
write('* Setup on repair: No;')nl,
retract(fact(frame(machine_info,A,no_operations,val,B))),
retract(fact(frame(machine_info,A,rep_time,val,C))),
retract(fact(frame(machine_info,A,scrap_part,val,D))).
```

the others are for when there is **no breakdown, a breakdown according to the time a machine is available or a breakdown according to the time a machine is busy**.

- There are 6 **machine _type** clauses, one for each type of machine, where for the batch machines it is:

```
mach_type(A,T):-T=batch->                              %test for batch machine
write('Batch;')nl,
fact(frame(machine_info,A,bat_max,val,C)), %retrieve frames with maximum and
fact(frame(machine_info,A,bat_min,val,D)),                minimum batch sizes
write('*Batch min:'),
write(C),write(';')nl,                                  %write minimum batch size
write('*Batch max:'),
write(D),write(';')nl,                                  %write maximum batch size
retract(fact(frame(machine_info,A,bat_max,val,C))),
retract(fact(frame(machine_info,A,bat_min,val,D))).
```

for the **Batch** machine it writes Batch followed by Batch min and Batch max, together with the respective parameters. The other clauses are when the machines are either assembly or production

- The **labour_2** predicate has 3 clause one for cycle labour or repair or both. The one for repair is:

```
labour(A,D):-member('1 Repairing themachine',D)-%test to see if labour required for
write('Labor:')nl,                                    repairing machine
write('Repair:Yes;')nl,                               %write Yes as entry for repair
write('END'));
```

2 PROMODEL Model Construction Rules

2.1 PROMODEL Part modelling rules

The first step in generating the routing table is to generate the column headings. This was achieved by the rule below:

```
routing_table_header:-
tell('Promodel.mod'),          %Setup output stream to file Promodel.mod for model
writing
write('ROUTING')nl,nl,
write('                                   Output Next       Condi-       Move'),nl,
write('Part   Location  Operation (min)  part   location   tion   Qty   time (min)'),nl,
write('----   --------  -------------    ----   --------   -------  ---  ---------'), nl,
load_files('M:\PROCSEQ.PL',[if(true)load_type(source),all_dynamic(true)]),
(go;nl).
```

The write predicates generate the following column headings:

```
                                  Output Next       Condi-       Move')
Part   Location  Operation (min)  part   location   tion   Qty   time (min)
----   --------  -------------    ----   --------   -------  ---  ---------
```

The portion of the CRMS data which is transformed into the routing table is of the form:

```
fact(frame(entry_point,'1',_,load / unload,val,'1')).
fact(frame(process_seq,'1',1,machine_no,val,'1')).
fact(frame(process_seq,'1',1,conveyor_no,val,['1','2','3'])).
fact(frame(process_seq,'1',1,proc_time,val,'3')).
fact(frame(process_seq,'1',1,setup_time,val,'4')).
fact(frame(process_seq,'1',2,machine_no,val,'2')).
fact(frame(process_seq,'1',2,conveyor_no,val,['4','5','6'])).
fact(frame(process_seq,'1',2,proc_time,val,'3')).
fact(frame(process_seq,'1',2,setup_time,val,'4')).
fact(frame(process_seq,'1',3,machine_no,val,'3')).
fact(frame(process_seq,'1',3,conveyor_no,val,['7','8','9'])).
fact(frame(process_seq,'1',3,proc_time,val,'3')).
fact(frame(process_seq,'1',3,setup_time,val,'4')).
fact(frame(exit_point,'1',_,load / unload,val,'1')).
fact(frame(process_seq,'1',2,conveyor_no,val,['10','11','12'])).
```

The rule for transforming this portion of CRMS data into the routing table is of the form:

```
[frame(entry_point,A,_,load/unload,val,B),
frame(process_seq,A,N,machine_no,val,C),
frame(process_seq,A,N,conveyor_no,val,D]),
frame(exit_point,A,_,load/unload,val,E)]
=>
[retract(all),     %retracts all data that matches LHS to prevent continuousexectiion of rule
tell('promodel.mod'),                    %setups file promodelmod for output
write('p'),                              %write p to denote part in part column
write(A),                                %write part number
write('     '),
write('load/unload'),                    %write load/unload in location column
write(B),                                %write load/unload station
write('      0'),                        %write 0 in operation column
write('      '),
write('p'),                              %write p to denote part in output part column
write(A),                                %write part number
write('      '),
write('m'),                          %write m to denote machine in next location column
write(C),                                %write first machine visit after entry
write('     0      1      conveyor'),   % write rest of first line if routing table
nl,
route(A),                                %action that writes the rest of the route of part
nl,                                         see below for explanation
write('p'),                              %write p to denote part in part column
write(A),                                %write part number
write('     '),
```

```
write('load/unload'),                    % write load/unload in location column
write(E),                                % write load/unload station for part exit
write('     0'),                         % write 0 in operation column
write('     '),
write('p'),                              % write p to denote part in output part column
write(A),                                % write part number
write('   Exit   0     1        conveyor)]   % write rest of last line if routing table
```

The rule writes load/unload station signifying the entry point of part, executes the user written action ***route_1*** and writes the last line containing the exit point for the part. The only user written action is ***route_1*** which has the structure:

```
route(A):-
fact(frame(process_seq,A,N,machine_no,val,C)),    % access the frames of visit
                                                  number N, machine number,
fact(frame(process_seq,A,N,conveyor_no,val,D)),   conveyor number to get there
fact(frame(process_seq,A,N,proc_time,val,E)),     and the processing time.
write('p'),                              % write p to denote part under part column
write(N),                                % write part number
write('....'),
write('m'),                              % write m to denote machine under location column
write(C),                                % write machine number
write('.....'),
write(E),                                % write processing time under operation coulmn
write('....'),
write('p'),                              % write p to denote part under output part column
write(N),                                % write part number
retract(fact(frame(process_seq,A,N,machine_no,val,C))),  % retract current mc visit
fact(frame(process_seq,A,N,machine_no,val,C))            %, access next machine visit
write('m'),                              % write m to denote machine under next location column
write(C),                                % write machine number
write('.....'),
write('     0   1   CONVEYOR'),nl,       % signifies conveyor system used for transport
route_end(A).                                                  % test for route end
route_end(A):-
frame(process_seq,A,N,machine_no,val,C),  % see if frame entry exists for next part
route(A).                                 % if yes redo action route(A) otherwise the
                                          complete route of part A has been written
```

This rule would generate the following entries in the routing table in the case of the above data:

Part	Location	Operation (min)	Output part	Next location	Condition	Qty	Move time (min)
p1	load/unload1	0	p1	m1	0	1	conveyor
p1	m1	3	p1	m2	0	1	conveyor
p1	m2	3	p1	m3	0	1	conveyor
p1	m3	3	p1	load/unload1	0	1	conveyor
p1	load/unload1	0	p1	Exit	0	1	conveyor

As with the routing table the first step in writing the part scheduling table is to generate the column headings, and is done in way similar to that for the routing table.

The portion of CRMS data which is translated into the part scheduling table is of the form:

```
[frame(entry_point,A,load / unload,val,1),
frame(creation_info,1,lot_size,val,2)
frame(creation_info,1,first_arr_tim,val,3)
frame(creation_info,1,max_arr,val,4)
fact(frame(creation_info,'1',inter_arr_tim,val,5)).
```

The rule for doing this is of the form:

```
[rule 4#:
frame(entry_point,A,load / unload,val,A),
frame(creation_info,A,lot_size,val,B),
frame(creation_info,A,inter_arr_time,val,C),
frame(creation_info,A,max_arr,val,D),
frame(creation_info,A,start_time,val,E)]
    =>
[retract(all),
tell('promodel.mod'),
write('P'),                    %write P to denote part under Part column
write(A),                      %write part number
write('      '),
write('load/unload'),  %write load/unload to denote part under location column
write(F),                          %write load/unload station number
write('          '),
write(B),                              %write lot size
write('          '),
write(D),                          %write maximum arrivals
write('          '),
write(E),                              %first arrival time
```

```
write('        '),
write(C),nl].                    %write inter-arrival time
```

The CRMS data shown above would be translated into the part scheduling module as:

Part	Location	Qty per arrival	No. of arrivals	Start (min)	Arrival frequency (min)
p1	load/unload1	2	4	3	5

2.2 PROMODEL Machine Modelling Rules

In PROMODEL machines are defined in the capacities table. The column headings for the capacities table are generated using the rule:

```
capacities:-
tell('Promodel.mod'),          %Setup output stream to file Promodel.mod for model
writing
write('CAPACITIES'),nl,nl,
write('RESOURCE    QTY'),nl,
write('--------------    -------    '),nl,
load_files('M:\PROCSEQ.PL',[if(true)load_type(source),all_dynamic(true)]),
(go;nl).
```

The rule for writing the capacities table entries is of the form

```
[frame(machine_info,X,quantity,val,A)]
    =>
[retract(all),
tell('promodel.mod'),
write('m'),                              %write m to denote machine
write(X),write('      '),                %write machine number
write(A),nl].                            %write machine quantity
```

Unlike in WITNESS, where a single rule is used to generate the machine characteristics, two rules are required in PROMODEL one for its definition within the capacities table and other for its breakdown and setup information within the downtimes table. Since the latter

rule has a similar structure to the rule for generating the capacities table entries its explanation is omitted for the purposes of conciseness.

3. FACTOR/AIM Model Construction Rules

3.1 FACTOR/AIM Part Modelling Rules

The first in generating the DEMANDnnn table is to generate the column headings, and this is created in a way similar to that for PROMODEL.

```
frame(creation_info,1,lot_size,val,2)
frame(creation_info,1,first_arr_tim,val,3)
frame(creation_info,1,max_arr,val,4)
fact(frame(creation_info,'1',inter_arr_tim,val,5)).
```

The rule for doing this is:

```
rule 4#:
[frame(creation_info,A,lot_size,val,B),
frame(creation_info,A,inter_arr_time,val,C),
frame(creation_info,A,max_arr,val,D),
frame(creation_info,A,start_time,val,E)]
    =>
[retract(all),
tell('demand.tab'),
write('P'),                    %write P to denote part under Part column
write(A),                      %write part number
write('    '),
write('    '),
write(E),                      %write first arrival time
write('    '),
write(C),                      %write inter-arrival time
write('    '),
write(B),                      %write load size
write('    '),
write(D),nl].                  %write ordersize
```

The CRMS data shown above would be translated into the DEMANDnnn table as:

DEMAND	FIRSTARR	INTERTIME	LOADSIZE	ORDSIZE
p1	3	5	2	4

The next stage is to specify the part route via the JOBSTEPnnn table. The table heading is generated in a similar way to that described before.

```
jobstep_tab_head:-
tell('jobstep.tab'),      %Setup output stream to file jobstep.tab for model writing
write('DEMAND     JSID   NEXTJSID    RESID1'),nl,
write('-----------     -----    -------------    --------'), nl,
 (go;nl).
```

this would result in the table heading:

DEMAND	JSID	NEXTJSID	RESID1

The CRMS entries which are transfered into the JOSTEPnnn table are of the form:

```
fact(frame(entry_point,'1',load / unload,val,'1')).
fact(frame(process_seq,'1',1,machine_no,val,'1')).
fact(frame(process_seq,'1',2,machine_no,val,'2')).
fact(frame(process_seq,'1',3,machine_no,val,'3')).
fact(frame(exit_point,'1',load / unload,val,'1')).
```

The rule for transfering this data into the JOBSTEPnnn is of the form:

```
[frame(entry_point,A,_,load/unload,val,B),
frame(exit_point,A,_,load/unload,val,C)]
=>
[retract(all),      %retracts all data that matches LHS to prevent continuous exectiion
                    of rule
tell('jobstep.tab'),                    %set up jobtep.tab for output
write('p'),                             %write p to denote part in DEMAND column
write(A),                               %write part number
write('js1'),                           %write js1 for 1st step in JSID column
write('     js2'),                      %write js2 for 2nd step in NEXTJSID column
write('     load/unload'),              %write load/unload in RESID column
write(B),                       %write load/unload station number of part entry
nl,
route(A),                               %action for generating route of part A
```

```
write('p'),                          %write p to denote part in DEMAND column
write(A),                            %write part number
write('    '),
write('js'),                         %write js for current step in JSID column
js(X)                                %retrieve current jobstep number
write(X),                            %write current jobstep number
incjs,                               %increment current jobstep
write('    '),
write('js'),                         %write js for last jobstep in NEXTJSID column
js(X)                                %retrieve current jobstep number
write(X),                            %write current jobstep number
write('    load/unload'),            %write load/unload in RESID column
write(C),                            %write load/unload part exit station number
nl]
```

The built in predicate ***route*** that transfers the CRMS route data into the JOBSTEPnnn table

is:

```
route(A):-
fact(frame(process_seq,A,N,machine_no,val,C)),   %access the frames of visit
                                                 number N,
write('p'),                          %write p to denote part in DEMAND column
write(A),                            %write part number
write('    '),
write('js'),                         %write js for current step in JSID column
js(X)                                %retrieve current jobstep number
write(X),                            %write current jobstep number
incjs,                               %increment current jobstep
write('    '),
write('js'),                         %write js for last jobstep in NEXTJSID column
js(X)                                %retrieve current jobstep number
write(X),                            %write current jobstep number
write('    m'),                      %write m to denote machine in RESID column
write(C),                            %write machine visit number
retract(fact(frame(process_seq,A,N,machine_no,val,C))),   % retract current machine
inc(N),                              %increment visit number
route_end(A,N).                      %test for route end


route_end(A,N):-
frame(process_seq,A,N,machine_no,val,C),  %see if frame entry exists for next vist
route(A).                            % if yes redo action route(A) otherwise the
                                     complete route of part A has been written
```

This rule would result in the followingJOBSTEPnnn table:

DEMAND	JSID	NEXTJSID	RESID1
p1	js1	js2	load/unload1
p1	js2	js3	m1
p1	js3	js4	m2
p1	js4	js5	m3
p1	js5	js6	load/unload1

3.2 FACTOR/AIM Machine Modeling Rules

In FACTOR/AIM machines are defined in the RESRCnnn table. The column headings for the RESRCnnn table are generated using the rule:

```
resrctabheading:-
tell(RESRC.tab'),                    %Setup output stream to file RESRC.tab for tab writing
write(' RESID          RESTYPE '),nl,
write('--------        ----------'),nl,
 (go;nl).
```

this results in the table headings:

The rule for detailing a machine is :

```
rule 2#:
[frame(machine_info,A,quantity,val,'1'),]                    %if '1' in value slot single
=>                                                           .. .it is a single machine
[retract(all),
tell(RESRC.tab'),
write('m'),                                          %write m to denote machine
write(A),                                                    %writes machine
number
write('      '),
write('machine')]                                    %write machine to
denote single                                                resource
type
```

The breakdown information of a machine is specified in the RESBRKnnn table. The column headings for the RESBRKnnn table are generated using the rule:

```
resbrktabheading:-
tell(RESBRK.tab'),    %Setup output stream to file RESBRKRC.tab for tab writing
write('VALUE  BETED FRSTED        REPAIRRESBRKID      RESID'),%write headings
nl,
write('-------  -------  --------        --------        ----------      -------'), nl,
(go;nl).
```

The rule generating the RESBRKnnn table is of the form:

```
rule 2#:
[frame(machine_info,A,breakdown_type,val,'2  According  to  the  time  the  machine  is
available').
frame(machine_info,A,first,val,B)).
frame(machine_info,A,rep_time,val,C)).
frame(machine_info,A,down_interval,val,D))]

=>
[retract(all),
tell(RESBRK.tab'),                    %setup RESBRK.tab as output file
write('onshift'),                          %write onshift for type
write('        '),
write(D),                             %write down interval
write('        '),
write(B),                             %write time of first break down
write('        '),
write(C),                             %write repair time
write('        '),
write('break'),                       %write break to denote breakdown
write(A),nl,                          %write breakdown number
write('m'),nl,                        %write m to denote machine
write(A)]                             %write machine number
```

Appendix D Running the system

1. Run 386-PROLOG version 2.3 for windows.
2. Loading the specification elicitation(CRMS) and the dialogue interface code.

   Consult file setup.pl by typing ***[setup]*** at the Prolog prompt **?-**. This consults: the specification elicitation files ***spec1.pl*** and ***spec2.pl***; and the PDM dialogue files ***winbox1.pl***, ***winbox2.pl***, ***winbox3.pl*** and ***winbox4.pl***. This also loads the forward chaining inference mechanism required for the model generation. In addition it also setups the output streams which will be used for writing the CRMS, WITNESS models, PROMODEL models and FACTOR/AIM models to file.
3. Run the Dialogue system by typing the Prolog query ***run***. This brings up the sequence of dialogues and writes the data entered to the file struc.pl(CRMS)
4. To generate a WITNESS list file corresponding to the CRMS consult file modconrul.pl, which setups all output streams for writing to files and consults the files containing the WTNESS ***define*** and ***detail*** section rules. To generate the WITNESS model type the query ***go***, which activates the forward chaining inference that matches the CRMS against the generation rules to generate the model. The generated model will be found in the file ***witness.lst***
5. To generate the PROMODEL models, FACTOR/AIM model and the English description consult the files promodegen.pl, factorgen.pl and langen.pl respectively. Again type the query ***go*** to start the generation.